

日本国特許庁
JAPAN PATENT OFFICE

IB04/03266

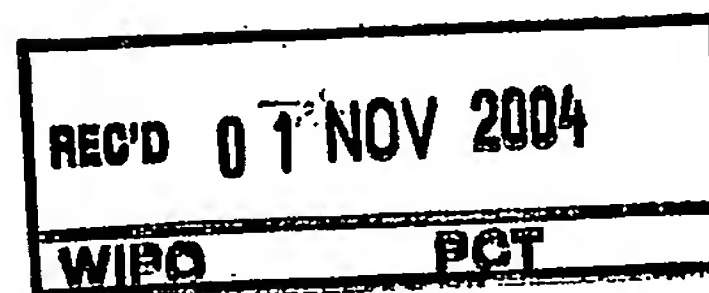
別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: 2003年10月 6日

出願番号
Application Number: 特願2003-346442
[ST. 10/C]: [JP2003-346442]

出願人
Applicant(s): カテナ株式会社

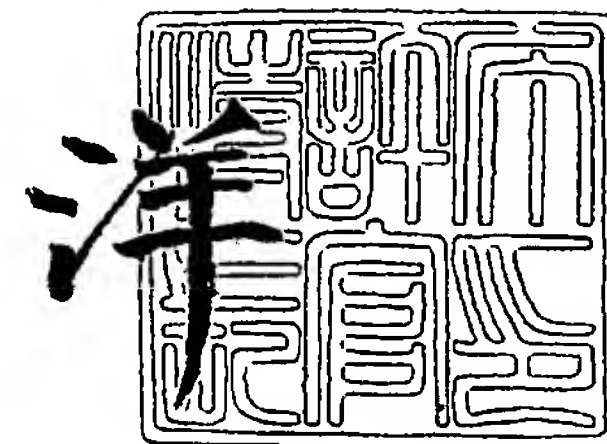


PRIORITY
DOCUMENT
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH RULE 17.1(a) OR (b)

2004年10月25日

特許庁長官
Commissioner,
Japan Patent Office

小川



出証番号 出証特2004-3096126

【書類名】 特許願
【整理番号】 DCT03045
【提出日】 平成15年10月 6日
【あて先】 特許庁長官殿
【国際特許分類】 F01B 23/12
F01C 17/00

【発明者】
【住所又は居所】 東京都多摩市落合 1 丁目 1 5 番 2 号 カテナ株式会社内
【氏名】 荒井 攻

【特許出願人】
【識別番号】 396023362
【氏名又は名称】 カテナ株式会社
【代表者】 小宮 善継

【代理人】
【識別番号】 100110559
【弁理士】
【氏名又は名称】 友野 英三

【手数料の表示】
【予納台帳番号】 164782
【納付金額】 21,000円

【提出物件の目録】
【物件名】 特許請求の範囲 1
【物件名】 明細書 1
【物件名】 図面 1
【物件名】 要約書 1

【書類名】 特許請求の範囲**【請求項 1】**

ソフトウェア開発要望に係る基本構造に属する単語ごとのエリアに値を求めるための単語単位プログラム相互の順序関係順序関係から第 1 の隣接行列を生成し、

前記基本構造同士の連結関係を規定する経路作用要素を単語とみなし、かかる単語の定義式及び実行条件を第 2 の隣接行列として表し、

前記第 1 及び第 2 の隣接行列からシステム全体を律する構造を成立させ、

前記成立した構造に対して探索を施して前記単語単位プログラム群を並び替える

ことを特徴とするソフトウェア開発前処理方法。

【請求項 2】

前記探索はトポロジカル・ソートによることを特徴とする請求項 2 記載のソフトウェア開発前処理方法。

【請求項 3】

ソフトウェア開発要望に係る単語ごとのエリアに値を求めるための単語単位プログラム相互の順序関係を有向グラフで表すことを特徴とするソフトウェア制御方法。

【請求項 4】

ソフトウェア開発要望に係る単語ごとのエリアに値を求めるための単語単位プログラム相互の順序関係に係る集合が閉じる単位で基本構造単位を構成し、

前記構成された基本構造単位をノードとして捉えた有向グラフで表すことを特徴とするソフトウェア制御方法。

【請求項 5】

ソフトウェア開発要望に係る単語ごとのエリアに値を求めるための単語単位プログラム相互の順序関係から隣接行列を生成し、

前記生成された隣接行列に対して探索を施し該探索の結果をもとに前記単語単位プログラム群を並び替え、

前記並び替えられた前記単語単位プログラム群を L y e e 方法論に適用してコードを生成する

ことを特徴とするソフトウェア開発方法。

【請求項 6】

ソフトウェア開発要望に係る単語ごとのエリアに値を求めるための単語単位プログラム相互の順序関係から隣接行列を生成する隣接行列計算部と、

前記生成された隣接行列に対して探索を施し該探索の結果をもとに前記単語単位プログラム群を並び替える探索部と、

前記並び替えられた前記単語単位プログラム群を L y e e 方法論に適用してコードを生成する生成部と

を具備することを特徴とするソフトウェア開発装置。

【書類名】 明細書

【発明の名称】 ソフトウェア開発前処理方法、ソフトウェア制御方法、ソフトウェア開発方法並びにソフトウェア開発装置

【技術分野】

【0001】

本発明は、ソフトウェア開発に係り、特に、数学的思考方にたったソフトウェア開発前処理方法、ソフトウェア制御方法、ソフトウェア開発方法並びにソフトウェア開発装置に関する。

【背景技術】

【0002】

一般にプログラムは、次のような3つの部分によって構成される。

- (1) 領域の宣言、メモリの割り当てとそれに関連した定数及び変数のアドレス
- (2) 制御の順序を表す文
- (3) メモリ割り当て領域に与えられる値を決定するための実行条件文と計算実行文

ソフトウェアが開発される時、ユーザは、主に(3)にのみ関係する。他の部分については、システムエンジニア及びプログラマが責任をもつ。人間は、コンピュータそれ自身を制御しているプログラムによって、画面及びキーボードからデータを入力し、ファイル中のデータを読み、またファイルにデータを書きこむ。

この部分に使用する装置を動かすプログラムも要件に依存せず、システムエンジニア及びプログラマが責任をもつ。このうち、特に、(2)の部分には要件に依存しない固定の(一定の)構造が存在する。したがって、要件に依存する(3)を決定する(すなわち、定義する)だけで、自動的にプログラムを生成することができる。

この構造体は L y e e (登録商標) の方法論の上では「プログラムの構造」といわれる(特許文献1乃至6、非特許文献3及び4参照)。

我々は通常、ホア論理 (Hoar Logic) や λ 計算のような論理的な関係を用いてプログラム構造を表現する(非特許文献1及び2参照)。このスタイルのプログラム構造の表現は、それらのプログラムの行動的構造を理解することの助けになる。これにより、特に宣言型(スタイル)プログラムの場合に、プログラムを理解し、分析することが可能になる。しかしながらこのようなアプローチを通してされた分析は、論理表現の持つ複雑さゆえに、意味論を検証したり、そして/あるいは、意味論を修正したりすることの困難さを顕在化する。

【特許文献1】 国際公開第97/16784号パンフレット

【特許文献2】 国際公開第98/19232号パンフレット

【特許文献3】 国際公開第99/49387号パンフレット

【特許文献4】 国際公開第00/79385号パンフレット

【特許文献5】 国際公開第02/42904号パンフレット

【特許文献6】 特開2002-202883号公報

【非特許文献1】 グリン・ウィンスケル著、「プログラミング言語の公式意味論」、MITプレス、1993年

【非特許文献2】 ハンネ・リース・ニールソン、フレミング・ニールソン著、「アプリケーションについての 意味論 公式紹介」、ジョン・ワイリー・アンド・サンズ、1992年

【非特許文献3】 根来 文生著、「Lyee ソフトウェアの原理」、21世紀 (IS2000) における情報社会についての国際会議会報、日本、会津、2000年11月5日-8日、pp441 - 446

【非特許文献4】 根来 文生著、「ソースコード生成についての高密度処理方法」、システミクス、サイバネティクス及びインフォマティクスについての第5回世界多極会議会報、(SCI2001)、米国、オーランド、2001年7月22日-25日

【発明の開示】

【発明が解決しようとする課題】

【0003】

このように、従来の論理的な関係を用いたプログラム構造についての表現は、特に宣言型(スタイル)プログラムの場合に、プログラムを理解し、分析することが可能になる。しかしながらこのようなアプローチを通してされた分析は、論理表現の持つ複雑さゆえに、意味論を検証したり、そして／あるいは、意味論を修正したりすることの困難さを顕在化するものであった。

【0004】

本発明は、前述のような従来の技術が有していた諸問題点を解決しようとするものであって、当該プログラムが複雑な論理表現を持っていたとしても、意味論を容易に検証し、修正し得る方法論を提供することを目的とするものである。

【0005】

本発明は、これにより、L y e e (登録商標) 方法論によるソフトウェア開発において、より高効率、高速、高パフォーマンスを実現することを目的とする。

【0006】

さらに本発明は、上記のような目的を達成できるソフトウェア開発前処理方法、ソフトウェア制御方法、ソフトウェア開発方法並びにソフトウェア開発装置を提供することを目的とする。

【0007】

また、本発明は上記と併せて、L y e e (登録商標) 方法論に係るプログラム構造を数学的に表現し、その正当性を示すこと、及び、その順序性のない性質を示すことを目的とする。

【課題を解決するための手段】

【0008】

上記課題を解決するために、本発明は、ソフトウェア開発要望に係る基本構造に属する単語ごとのエリアに値を求めるための単語単位プログラム相互の順序関係順序関係から第1の隣接行列を生成し、該基本構造同士の連結関係を規定する経路作用要素を単語とみなし、かかる単語の定義式及び実行条件を第2の隣接行列として表し、該第1及び第2の隣接行列からシステム全体を律する構造を成立させ、該成立した構造に対して探索を施して前記単語単位プログラム群を並び替えることを特徴とする。

【0009】

さらに本発明は、ソフトウェア開発要望に係る単語ごとのエリアに値を求めるための単語単位プログラム相互の順序関係を有向グラフで表し、該表された有向グラフに対して探索を施し、該探索の結果をもとに当該単語単位プログラム群を並び替えることを特徴とする。

【0010】

また本発明は、ソフトウェア開発要望に係る単語ごとのエリアに値を求めるための単語単位プログラム相互の順序関係から隣接行列を生成し、該生成された隣接行列に対して探索を施し、該探索の結果をもとに当該単語単位プログラム群を並び替えることを特徴とする。

【0011】

ここで、「探索」とは、(後述する) トポロジカル・ソート、(Topological Sort)、(後述する) 深さ優先探索 (Depth-First Search)、幅優先探索 (Breadth-First Search)、反復深化法 (Iterative Deepening)、発見的探索 (Heuristic Search)、ヒル・クライミング (Hill Climbing)、最適優先探索 (Best-First Search)、オイラー (Euler) の一筆書き、ダイクストラ (Dijkstra) 法等を含む各種アルゴリズムをいう。

【0012】

また本発明は、ソフトウェア開発のユーザ要件中の単語についての有意性規定関係から隣接行列及び状態ベクトルを生成し、当該隣接行列と状態ベクトルとを所定の公理に基づき計算し、該計算の結果から、単語の状態を把握することを特徴とする。

【0013】

さらに本発明は、ソフトウェア開発要望に係る単語ごとのエリアに値を求めるための単語単位プログラム相互の順序関係を有向グラフで表すことを特徴とする。

【0014】

ここで、順序関係としては半順序及び全順序が考えられる。

【0015】

また、ここで用いる「ソフトウェア」とは広義の意である。即ち、ユーザからの開発要望及び当該要望にあった目的プログラムの双方を包含する概念である。

【0016】

また本発明は、ソフトウェア開発要望に係る単語ごとのエリアに値を求めるための単語単位プログラム相互の順序関係に係る集合が閉じる単位で基本構造単位を構成し、該構成された基本構造単位をノードとして捉えた有向グラフで表すことを特徴とする。

【0017】

さらに本発明は、ソフトウェア開発要望に係る単語ごとのエリアに値を求めるための単語単位プログラム相互の順序関係から隣接行列を生成し、該生成された隣接行列に対して探索を施し該探索の結果をもとに当該単語単位プログラム群を並び替え、該並び替えられた当該単語単位プログラム群をL y e e方法論に適用してコードを生成することで求めるソフトウェアを得ることを特徴とする。

【0018】

また本発明は、ソフトウェア開発要望に係る単語ごとのエリアに値を求めるための単語単位プログラム相互の順序関係から隣接行列を生成する隣接行列計算部と、該生成された隣接行列に対して探索を施し該探索の結果をもとに当該単語単位プログラム群を並び替える探索部と、該並び替えられた当該単語単位プログラム群をL y e e方法論に適用してコードを生成する生成部とを具備することを特徴とする。これら各部は総て自動化できるので、結局繰返し処理の排除→ソフトウェア生成まで一貫して自動処理が可能となり、ソフトウェア生産速度、効率は各段に向上する。

【0019】

ここで、(後述する)「有向グラフ作成部」とは、ソフトウェア開発要望者であるユーザの発する要件に含まれるそれぞれの単語をノードによって表し、入力単語と状態単語とを区別しつつ $a \leq b$ (aからbが作られることを意味する記号として「 \leq 」を定義する。以下同じ。)という始点と終点の間の半順序関係(または全順序関係)を矢印によって表した有向グラフを作成する機能を実現する単位をいう。例えば、こうした機能をプログラム化したソフトウェアとしてもよいし、或いは当該ソフトウェアをコード化したものをROM (Read Only Memory) に書きこんだものであってもよい。

【0020】

ここで、「入力単語」とは(後述の)数式3における f_i を持っていない最終の終点単語をいい、「状態単語」とは入力単語でない単語をいう。なお、以下では、数式を指標するにあたり、数式の見出し中では「数」の表題に番号を付したものをもってするが、文章中指標するにあたっては、「数式1」のようにいうこともある。

【0021】

また、「隣接行列計算部」とは、単語単位のプログラム相互の関係を(後述の)数式3形式で表現し、それを行列の形式で再表現した形態である隣接行列を計算して得る機能を実現する単位をいう。例えば、こうした機能をプログラム化したソフトウェアとしてもよいし、或いは当該ソフトウェアをコード化したものをROM (Read Only Memory) に書きこんだものであってもよい。

【0022】

さらに、「探索部」とは、前述の各種手法を包含するアルゴリズムである探索を一般に

実現する機能を有する。例えば、こうした機能をプログラム化したソフトウェアとしてもよいし、或いは当該ソフトウェアをコード化したものをROM (Read Only Memory) に書きこんだものであってもよい。

【0023】

また、(後述の)「トポロジカル・ソート部」とは、上記で得られた有向グラフに対して(後述の)深さ優先探索を行い、行きついたところから探索経路を戻るときにノードを取得してゆくことでノード列(数列)を整列する機能を実現する単位をいう。例えば、こうした機能をプログラム化したソフトウェアとしてもよいし、或いは当該ソフトウェアをコード化したものをROM (Read Only Memory) に書きこんだものであってもよい。

【0024】

ここで、「深さ優先探索」とは、始点であるノードを出発してノード間の連結がとれているノードを訪問が重複しないように進み、行く場所がなくなったら行く場所のあるノードまで戻り(再帰)、戻ったノードから再び(1)の要領で進み、進むところがなくなったら終了する、というアルゴリズムをいう。

【0025】

上記で規定される本願に係る発明では、繰返しを回避するので、より高効率、高速、高パフォーマンスを備えたソフトウェア開発を実現することが可能となる。

【0026】

さらに上記で規定される本発明では、繰返しをなくす前処理を自動で行い、こうして得られた前処理済の単語単位のプログラムをもとにL y e e (登録商標)方法論により自動的に目的プログラム生成を行う。つまり、ユーザ要件の洗練化(整列)から目的プログラムの生成に至るまで自動化することが可能となる。これにより、ソフトウェア生産の大幅な効率向上、生産性向上、品質向上等、ソフトウェア産業上に大きな効果をもたらす。

【発明の効果】

【0027】

本発明によれば、L y e e (登録商標)方法論によるソフトウェア開発において、より高効率、高速、高パフォーマンスを実現することができる。

【発明を実施するための最良の形態】

【0028】

以下、図面を参照しながら、本願の発明の具体的な実施形態について説明する。

【0029】

第I章で、本発明の前提となる、技術および概念について述べる。第I章のI-1では、本発明が改良を施す対象であるL y e e (登録商標)方法論によるソフトウェア構造(単語単位のプログラム構造)と、L y e e (登録商標)の特徴的な概念である「状態」の概念について概略を述べる。I-2では、単語単位のプログラムを数学的構造モデルによって説明する。I-3で、単語単位のプログラムによる値の生成を一巡処理で完了するための技術トポロジカルソートについて述べる。トポロジカルソート技術については、PCT/J P 03/09591で特許出願済みである。

【0030】

第II章で、本発明の具体的な実施形態である、トポロジカルソートをL y e e 方法論のシステム全体に施すためのシステム構造の単一化について述べる。第II章のII-1では、L y e e のシステム構造について述べる。II-2で、L y e e のシステム構造の単一化技術について述べる。

I 章

I-1. 単語単位のプログラムとは何か

I - 1 - (1) 単語単位のプログラムとは

L y e e (登録商標) 方法論によるソフトウェアは、単語単位のプログラムを基本にして構成されている。単語単位のプログラミングでは、要件は単語の集合によって表され、単語は宣言文を使って定義される。単語単位のプログラムは、2種類の主なコンポーネントで構成されている。1つのコンポーネントは単語のコンポーネントである。単語単位のプログラミングでは、要件は、単語の名前、単語の計算式の定義、単語の計算式の条件、単語の属性（入出力、データタイプ、その他の属性）を含んだ単語の集合とし、代入文ではなく宣言文の形式で定義される。

たとえば、 S_w が単語 w を定義する宣言文であるとする、その要件の例は表 1 のようになる。

【0 0 3 1】

【表 1】

	単語名	属性	条件	定義
単語	a	出力	d	$b + c$
単語	c	入力 整数		
単語	b	出力	$c > 0$	$2 \times c + 5$
単語	d	入力 ブール		

もう 1 つのコンポーネントは制御プログラムのコンポーネントである。制御プログラムは、それ以上実行を繰返しても、いかなる単語の値も変化しない状態になるまで、単純に単語単位プログラムの実行の繰返しを行う。単語 a の定義は単語 b を使用しており、単語 b は単語 a の後に与えられているが、ユーザはこれらの定義が実行される順序（制御ロジック）を指定する必要がない。一方、手続き型プログラミングの場合は当然ながら、宣言型プログラミングの一種であるロジカルプログラミングの場合でも、プログラムの一行の実行順序は実行結果に影響を与える。

【0 0 3 2】

表 1 の例を使って、この処理の動きを説明する。

【0 0 3 3】

最初に、単語 a が実行される。単語 d 、 b 、 c はまだ実行されておらず、従って単語 a は値を決定することができない。次に、入力単語 c が実行される。単語 c は実行条件がないので、値が決定される。例えば、単語 c の値が $+3$ だとすると、単語 b は、実行条件 $c > 0$ が真となるので、定義 $2 \times c + 5$ が実行可能になり、単語 b の値は $+11$ と計算され、値の出力が可能となる。単語 d は入力単語であり、実行条件がないので、 $d = \text{真}$ である。これですべての単語が処理されたが、単語 a の値は未決定のままである。結果として、プログラムは最初に戻って再び処理を実行する。

【0034】

2 回目の実行時、d = 真、単語 b と単語 c はそれぞれ値が +11 と +3 に決定しているので、単語 a は値 +14 を得て、その値が適切な装置に出力される。単語 c、b と d の値は前に計算されているので、これらの単語の値のすべては決定済みである。従って、2 回目の実行で、4 つの単語の総ての値が決定したことになる。

【0035】

次に、「状態」概念について説明する。単語のデータタイプは、コンピュータ上で扱い可能な、数、ブール (boolean)、文字、その他である。論理的順序を制御するためには、それぞれの単語に存在する「状態」の概念の導入が必要となる。実際、単語の状態とは、複数の状態：すなわち、真、偽、そして未決定のいずれか 1 つとなる。それぞれの意味は次の通りである。

真：値が決定され、かつユーザ要件を満たす。

偽：値が決定されるが、それはユーザ要件を満たさない。

未決定 (又は空)：値が決定されていない。

単語の最初の状態は常に未決定で、それは単語の値がまだ転送 (move) されていないか、あるいは計算されていなかったことを意味している。名前によって定義された単語を終点単語、そして、別の単語を定義するために、計算条件や計算式に使われている単語群を始点単語と呼ぶ。要件中の総ての単語が 1 回実行されることを一巡処理と呼ぶ。

【0036】

単語の値が決定され、その状態が真となるのは、その単語の始点単語の状態がすべて真となったときである。始点単語として使われていない単語の状態は、終点単語の状態に関与しない。一巡処理が繰り返されることを、繰返し処理 (又は、繰返しプロセス) と呼ぶ。単語の状態に変化がなくなるまで繰返し処理を続けることにより、単語を出力に用いることが可能となる。各単語の状態は、真、偽になるか、あるいは未決定のまま残るか、のいずれかである。もし未決定の単語があるなら、そのシステムに入力単語が存在するかどうか、閉じたループがあるかどうかをチェックする必要がある。

【0037】

閉じたループは、単語が直接的または間接的に自身で自身を定義しているか、要件が完全でないことを意味する。実行条件は排他的でなければならず、また否定は明示的に定義されていなくてはならない。すなわち、条件の総和が重なりのない全体を構成 (言い換えれば、条件の総和が 1 となり、1 以上でも 1 以下でもない) しなければならない。

【0038】

このように、単語単位のプログラミングは、ソフトウェアの制御ロジック部分を自動的に生成する。

I-1-(2) 個々の単語のプログラム構造

図 1 は、前述した単語単位のプログラムの構造を示したフローチャートである。同図をもとにプログラムの構造及び動作について説明する。図 1 の α は、単語名を示す。

(1) ボックス 1

まず、statement (宣言文) を遂行すべきかどうか判断する。もし単語 α の状態が空でないなら、以下を実行せず終了し (301)、空ならボックス 2 へ進む。

(2) ボックス 2

もし始点単語の状態が総て真であるときは、単語の定義式 (計算式) を遂行して、その結果を一時的な領域に記憶する。従って、一時的な領域が必要となる。始点のいずれかが真以外の場合は、定義式を遂行しない。

(3) ボックス 3

定義式を実行した、または不実行の結果、単語 α に有意な (ユーザ要件に適合する) 値が決定されたかどうかを判断する。総ての始点単語が真であるとき、単語の値は決定され

る。しかし、たとえ総ての始点単語が真であるとしても、ユーザの要求条件によっては単語の値は偽とされるかもしれない。単語に有意な値が決定されるときはボックス 4 へ、単語に値が決定されない、又は決定された値が偽とされるときはボックス 5 へ移行する。

(4) ボックス 4

決定された単語 α の値がユーザの要求条件に適合するときには、通常領域に結果を記録する。従って、通常領域が必要となる。この場合には、単語 α の状態は真である (3 0 3)。

(5) ボックス 5

ボックス 3 で単語の値が未決定、又は決定された値が偽とされた場合はここに至る。ボックス 5 では、単語がこのうちどちらであるかを判断する。

単語の値がユーザ要求条件によって偽となるか、あるいは始点の値が偽の場合は、単語の状態は偽である。単語の状態が偽となる場合はボックス 6 へ、単語の値が偽でない場合はボックス 7 へ移行する。

(6) ボックス 6

値が偽であることを示すフラグをセットする。このようなフラグは偽のプロセスとしてすべての単語に必要な。または、状態が偽であることを意味する値を通常領域に記憶する。このとき単語の状態は偽となる (3 0 5)。

(7) ボックス 7

繰り返し処理を実行するかどうかを判断するために再帰 (繰返し) カウンタに 1 を加える。従って、再帰カウンタが必要となる (非特許文献 3 参照)。このとき単語の状態は空となる (3 0 7)。

I - 1 - (3) 全体のシステムを制御するプログラムとその構造

システム全体を制御するプログラムがパレット連鎖関数と呼ばれる (非特許文献 3 参照)。適切に繰り返し処理をする制御プログラムはこのプログラムの中に入れられる。図 1 で説明したように、もし 1 つの始点単語の状態が偽であるなら、終点単語の状態が確実に偽になる。たとえ始点単語のすべての状態が真であるとしても、ユーザ要求条件について偽があるときには、終点単語の状態が偽になる。したがって、繰り返しがなされるかどうかを判断するためには、ただ空 (未決定) の始点単語が定義式に含まれるか否かを判断すればよい。

【0 0 3 9】

もし空の始点単語が 1 つでもあれば、繰り返し処理は実行される。しかしながら、有向グラフによってそれを表現するならば、閉じられたループが含まれるときには、終点単語の状態が空から変化しない。また、このシステムが始点単語を持っていないとき、終点単語は空から変化しない。そのために、このような場合には、繰り返しプロセスは終了せず永久のループに入る。このような状況を回避するために、プログラムのすべての単語 $L(\alpha)$ を制御するパレット連鎖関数のプログラム (Φ) にある再帰カウンタを制御するために用いられるプログラムコードがある。

【0 0 4 0】

単語が空になるたびに、それぞれの単語のプログラムが再帰カウンタに 1 を加える。一連の繰り返しプロセスを終了する条件として、この再帰カウンタの値が前回の繰り返し処理での値と同じであることを用いる。この状態を「状態変化はない」という。

【0 0 4 1】

ロジカルプログラミングでは、特に非手続き型プログラミング言語の一種である標準的なプロログ (Prolog: 非手続き型プログラミング言語) では、バックトラック (再実行) が永久ループに陥る。他の改良型プロログは、待機 (wait) やフリーズを使って永久ループを回避する。単語単位のプログラムでは、前述のメカニズムを使ってこのようなループを回避する。

L y e e (登録商標) 方法論で用いるパレット連鎖関数プログラム (Φ) の例を以下に

示す。「Crtの値は状態が未決定（空）である単語 $L(\alpha)$ の数である」をループの変数プロパティとするというプロパティを用いることによって、このプログラムはホア（Hoare）の理論で証明することができる。下記の例は x 、 y 及び z の 3 つの単語がある場合である。

[Φ のプログラムコード]

```
PCtr = 1      /* PCtr = 前回の繰り返し処理の再起カウンタ */
```

```
Ctr =
```

```
0      /* Ctr = 再起カウンタ */
```

```
While
```

```
PCtr &supl; Ctr do
```

```
    PCtr = Ctr
```

```
    Ctr = 0
```

```
L(x)      /* 各単語プログラムは状態と繰り返しの制御を行う以下のような関数（機能）を持っている;
```

```
もし始点単語が空なら、単語 $L(\alpha)$ の状態は未決定で Ctr = Ctr+1
```

```
そうでないなら、単語 $L(\alpha)$ の状態は決定済*/
```

```
L(y)
```

```
L(z)
```

```
End
```

```
/* Ctrの値は状態が未決定である単語 $L(\alpha)$ の数*/
```

このプログラムは以下の意味になる：

プログラムが終了する。

終了したとき、変数Ctrの値は、状態が未決定である単語 $L(\alpha)$ の数である。

プログラムが停止したとき、単語の定義式に従って、このシステム中に始点単語を持つ単語 $L(\alpha)$ の状態は、通常の場合真か偽が割り当てられる。単語 α の状態は、次のような場合は、未決定（空）である：

1) 単語 α が閉じたループの中か、閉じたループの後にある場合。

2) 単語 α が排他的条件を持っていない場合。

排他的とは、もし

【0 0 4 2】

【数1】

$((x \geq y) \text{ or } (x < y))$

が常に真であり、かつ、

【0 0 4 3】

【数2】

$((x \geq y \text{ and } (x < y))$

が常に偽であれば、 x

と y

は排他的である、として定義される。

3) 単語 α の始点単語が同じシステムにない場合。

I - 2. 単語単位のプログラムの数学的考察

単語単位のプログラムのプログラム構造はホアの定理によって証明できる。しかしながら、そのようなホアの定理のアプローチを通しての分析では、その論理表現のスタイルの複雑性のために、単語単位プログラムの意味論を検証し、改良を加えることが困難である

。そこで、有向グラフ技術に起源したもう 1 つのアプローチである隣接行列を用いて、単語単位プログラムによるプログラム構造を数学的構造モデル化し、本発明を説明する。この章では、まず単語単位のプログラムによるプログラム構造を、数学的構造モデルによって表し、その特徴を述べる。すなわち、有向グラフモデルと隣接行列モデルである。

I - 2 - (1) 単語単位プログラムの関数表現

Lyee 要件の仕様をより形式的に扱うと、関数 x を、1、2、 \dots 、 n の点における定数とその他の点の x の値によって定義した方程式のシステムに表すことができる。

【0 0 4 4】

【数 3】

$$x(i) = f_i (x(1), x(2), \dots, x(n)) \quad i = 1, 2, \dots, n$$

数式 3 は、単語が n 個あるシステムにおける、 i 番目の単語 x の定義式を表す関数である。 $x(i)$ は終点単語である i 番目の単語 x を表す。 $f_i (x(1), x(2), \dots, x(n))$ は、終点単語 $x(i)$ の定義式を表す関数で、定義式がシステム内の 1 から n 番目の単語のいずれかによって構成されていることを示す。

数式 3 の Lyee (登録商標) 仕様は次のように書き直すことができる：

【0 0 4 5】

【数 4】

$$x(i) = F (x(i)) \quad i = 1, 2, \dots, n$$

さらに簡易な形式では、
 $x = F (x)$
 である。

I - 2 - (2) 本発明の原理：有向グラフ (directed graph) とその隣接行列

始点単語と終点単語の関係は、有向グラフによって表すことができる。有向グラフでは、各単語はノードによって、単語間の関係は矢印によって表される。矢印の出発点となるノードが始点単語、終着点となるノードが終点単語を表す。さらに、それは $a \leq b$ (単語 a から単語 b が作られる) という関係をもつ順序を定義する。

【0 0 4 6】

図 2 は、 $a \leq b$ (a から b が作られる) という関係 (f) をもつ順序を説明するための有向グラフの例を示した図である。

【0 0 4 7】

同図に示すように、単語は $a \leq b$ という関係を持つ半順序づけられた集合 (セット) を構成する。即ち、同図の有向グラフは単語 a 及び単語 b という要素及びこれらの要素間を関係付ける半順序関係 (矢印) を構成要素とする集合とみることができる。

【0 0 4 8】

有向グラフは隣接行列によって表現することができる。 n 個の単語の関係を表す隣接行列は、 n 個 \times n 個のマトリックスで表され、これを隣接行列 F と呼ぶ。この隣接行列において、行見出しと列見出しには、それぞれ n 個の単語名が置かれる。行の見出しとなるとき n 個の単語はそれぞれ終点単語であり、列の見出しとなるとき n 個の単語はそれぞれ始点単語である。

【0 0 4 9】

i 番目の単語の式 (つまり、数式 3 によれば、単語 $x(i)$ を生成するプログラム $f_i(x(i))$) が、始点単語として x_j を使うとき、隣接行列の i 行目と始点単語 x_j 列の交点を表す隣接行列 F の要素 f_{ij} ($i=1, \dots, n; j=1, \dots, n$) は 1 である。そうでないなら、0 である。終点単語 x_i を生成する式において、始点単語 x_j が使用されているか否かを 1 か 0 で表すのである。(※「 f_{ij} ($i=1, \dots, n; j=1, \dots, n$)」は、単語 $x(i)$ を生成するプログラム $f_i(x(i))$ を構成する単語のうちの単語 $x(j)$ を指し、このときは i は 1 から n 、 j は 1 から n のいずれかである、ことを示している。)

各単語が数式 3 の形式で、下記の数式 5 のように定義される例 1 の単語の関係を、有向グラフ、隣接行列の 2 つの形態で表して具体的に説明する。

【実施例】

【0050】

(例 1)

入力単語 ($f_i(x(i))$ を持っていない最終の終点単語) を x_1 及び x_2 と置き、状態単語 (入力単語でない単語、すなわちプログラムによって値を生成する単語) を x_3 、 x_4 、 x_5 、 x_6 及び x_7 と置く。これらの関係を上記数式 3 の形式で表すと下記の数式 5 のようになる。

【0051】

【数 5】

$$x(1) = a_0$$

$$x(2) = b_0$$

$$x(3) = f_3(x(4), x(5))$$

$$x(4) = f_4(x(2), x(5))$$

$$x(5) = f_5(x(1))$$

$$x(6) = f_6(x(3), x(7))$$

$$x(7) = f_7(x(4), x(6))$$

図 3 は、上記の例 1 の状態を表した有向グラフの例である。

【0052】

即ち、同図に示す例においては、 x_1 及び x_2 は $f_i(x(i))$ を持っていない入力単語である。よって、いずれの矢印の終着点にもなっていない。 x_3 は x_4 及び x_5 から生成される。よって、 x_3 は x_4 及び x_5 から出発する矢印の終着点になっている。同様に、 x_4 は x_2 及び x_5 から、 x_5 は x_1 から、 x_6 は x_3 及び x_7 から、 x_7 は x_4 及び x_6 から生成されることを表すように矢印が出发点と終着点が表されている。つまり、数式 5 で規定される状態と図 3 の有向グラフで表される状態とは等価である。

【0053】

このとき例 1 の隣接行列は数式 6 のように表現される。

【0054】

【数 6】

$$F = \begin{matrix} & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 & \mathbf{x}_5 & \mathbf{x}_6 & \mathbf{x}_7 \\ \mathbf{x}_1 & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{x}_2 & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{x}_3 & \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \\ \mathbf{x}_4 & \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \\ \mathbf{x}_5 & \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{x}_6 & \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \\ \mathbf{x}_7 & \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

先に述べたように、隣接行列 F の行見出しは終点単語であり、列見出しは始点単語である。各行または列がどの終点単語または始点単語を示すかをわかりやすくし、説明を簡便にするために、隣接行列には見出しを付した。

【0 0 5 5】

数式 6 の隣接行列 F を各行について展開して説明する。第 1 行目の終点単語 x_1 の行は、いずれの始点単語列との交点も 0 になっており、終点単語 x_1 は始点を持たないこと（すなわち入力単語であること）が示されている。終点単語 x_2 についても同様である。第 3 行目の終点単語 x_3 の行は始点単語 x_4 と x_5 の列との交点が 1、その他の始点単語列との交点が 0 になっており、終点単語 x_3 は x_4 と x_5 を始点単語として持つことが示されている。以下、終点単語 x_4 から終点単語 x_7 の終点単語についても、同様に、どの単語を始点単語として持つか（あるいは持たないか）が示されている。

【0 0 5 6】

数式 6 の隣接行列の各行は、以下のような数式でも表すことができる。左辺が行見出しとなる終点単語、右辺は、交点の値が 1 となった列見出しの始点単語を表す。右辺の「0」は、いずれの交点の交点の値も「0」であることを表す。

$$x_1 = 0$$

$$x_2 = 0$$

$$x_3 = x_4$$

$$+ x_5$$

$$x_4 = x_2 + x_5$$

$$x_5 = x_1$$

$$x_6 = x_3 + x_7$$

$$x_7 = x_4 + x_6$$

ここで、「+」は、後述するように、通常 of 四則演算における「加える」の意味でなく、本明細書中の単語の関係を表す数式においては、左辺の状態単語（終点単語）を決める半順序集合の始点単語が複数あるときにこれらを付加して左辺の単語を規定することを示す演算子（以下、「付加演算子」と呼ぶ）として用いている。

【0 0 5 7】

以上より、したがって、数式 5 で定義される単語の関係は、図 3 の有向グラフで表される関係及び数式 6 の隣接行列による数学的表現で記述される関係と等価であるということがいえる。

I - 2 - (3) 隣接行列 F と状態ベクトル X で表す終点単語のための状態モデル

前セクションでは、n 個の単語の「終点単語の他の単語との関係」を表すためのモデルとして、有向グラフと隣接行列を考察した。次に、n 個の単語の「終点単語の値の状態」を表すためのモデルについて述べる。まず、n 個の単語の始点単語としての状態を縦列のベクトル X に記録する（ベクトルとは 1 つの列あるいは行しか持たないマトリックスである）。数式 6 の隣接行列に表された単語の例で説明すると、始点単語が表示される順序、 x_1 から x_7 の順序に各始点単語の状態を表したベクトル X は、たとえば下記の数式 7 のように表すことができる。各行がどの始点単語を示すかをわかりやすくし、説明を簡便にするために、ベクトルには見出しを付した。

【0 0 5 8】

【数 7】

$$X = \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{matrix} \begin{bmatrix} +1 \\ +1 \\ -1 \\ \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \end{bmatrix}$$

このとき、ベクトル X は、始点単語 x_1 と x_2 の状態は「+1」、始点単語 x_3 の状態は「-1」、 x_4 から x_7 の始点単語の状態は「null」であることを示している。

【0 0 5 9】

単語を生成し得る値の状態は次のように 3 種類に設定される。

+ 1 : 値は真であるとして決定済みである。

- 1 : 値は偽であるとして決定済みである。

空(null) : 値は未決定である。

入力単語の初期値は以下の通りである。

1: 値は真であるとして決定済みである。

(適切に入力されている状態)

— 1: 値は偽であるとして決定済みである。

(適切に入力されていない状態)

従って、たとえ数式 3 が実行されるとしても、状態は変化しない。

【0060】

状態単語の初期値は空である:

空: 値はまだ決定済みでない。

(移動していない、あるいはまだ計算されていない状態)

状態単語について考察するに、単語の関係を表す隣接行列Fの上に単語の状態を表す状態ベクトルXを作用させるということは当該単語の順序で数式 3 を横列で実行することである。順番に n 個の式を使う処理は順序処理と呼ばれる。その始点単語のすべての状態が +1 であるとき、状態単語の状態が +1 (真) になり、状態単語の値は決定済みとなる。

【0061】

始点単語ではない単語の状態はその式の終点単語の状態とは関係していない。順序処理が繰り返されることは繰り返し処理と呼ばれる。

たとえそれが繰り返されるとしても、「真」であるとして決定が済んでいない単語は、偽の状態 (-1) になるか、あるいは空のままである。上記のことが「有向グラフ」で表現されると以下のようなになる。

(1) もしノードが閉じられたループに含まれなかったなら、そして、もしいずれかの始点単語に係るノードから次々に始点単語を探索した際に当該始点単語の値が偽であるなら、ノード (単語) の値は決定されない。もし単語の値がユーザの必要条件を満たさなかったならノードの値は決定されない。たとえ値が成功裏に計算されるとしても、ノード (単語) の値が偽と決定される。

(2) もしノードが閉じられたループに含まれなかったなら、そして、もしすべての始点単語が、そのノードから次々に始点単語を探索したとき真であるなら、ノードの値は真として決定される。

(3) ノードが閉じられたループに含まれるときであって、ノードの始点単語がシステムに存在しないとき、ノード (単語) の値は空のまま残る。

(例 2)

例 1 の場合、図 3 において、 x_3 、 x_4 、 x_5

、 x_6 及び x_7 の状態は、この場合における終端の始点単語である x_1 及び x_2 の状態に依存する。

同図から明らかなように、終端の始点単語である x_1 が真になるとき、 x_5 は真である。終端の始点単語である x_1 及び x_2 が真であるとき、 x_4 及び x_5 が真になる。 x_6 及び x_7 はループにあるので空のままである。

I-2-(4) 隣接行列Fと状態ベクトルXの計算操作

n 個の単語の状態はベクトルX (列 1 つからなる行列) に、単語の順序に従って記録される。隣接行列Fと状態ベクトルXを用いて繰り返し処理を実行して計算した状態と同じ状態を表すように、算術演算子が定義される。言い換えれば、FとXの乗算は以下のようなになる。

【0062】

【数 8】

$$X_m = FX_{m-1}$$

この時、

X_m, X_{m-1} : n 個の単語の列ベクトル
 X_0 : 単語の初期ベクトルの 全ての要素は空 (null)
 $x_{m,i}$ of X_m : (-1) or (+1) or (null)
 f_{ij} of F : 0 or 1
 n : 単語の個数 items of words
 m : 繰り返しの回数

m 回目の隣接行列 F と状態ベクトル X の積である、状態ベクトル X_m の要素 $x_{m,i}$ (m 回目の繰り返し処理後の単語 x_i) は以下のような式で表せる。

【0 0 6 3】

【数 9】

$$x_{mi} = f_{i1} \cdot x_{m1} + \dots + f_{i(i-1)} \cdot x_{m(i-1)} + f_{ii} \cdot x_{m,i} + f_{i(i+1)} \cdot x_{m,i+1} + \dots + f_{in} \cdot x_{m,n}$$

乗算演算子 (\cdot) の左側の f_{ij} には、隣接行列 F の i 行目の終点単語において j 列の始点単語 x_j が使われているか否かを表す指数 1 か 0 (始点単語となる場合が 1、始点単語とならない場合が 0) が与えられる。乗算演算子 (\cdot) の右側の $x_{m,i}$ には、 m 回目の繰り返し処理中のベクトル X の i 行目に表された始点単語 x_i の状態を示す 3 種類の指数 +1 (真)、-1 (偽)、null (ここで null は空を意味する) が与えられる。

【0 0 6 4】

数式 6 の隣接行列 F と数式 7 のベクトル X を例にして、終点単語 x_3 の 3 回目の繰り返し処理を数式 9 で表すと以下ようになる。

$$\begin{aligned}
 x_{3,3} &= f_{3,1} \cdot x_{3,1} + f_{3,2} \cdot x_{3,2} + f_{3,3} \cdot x_{2,3} + f_{3,4} \cdot x_{2,4} \\
 &+ f_{3,5} \cdot x_{2,5} + f_{3,6} \cdot x_{2,6} + f_{3,7} \cdot x_{2,7} \\
 x_{3,3} &= 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (\text{null}) \\
 &+ 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot (\text{null}) + 0 \cdot (\text{null})
 \end{aligned}$$

数式 6 の y_3 行と始点単語 x_1 列との交点は「0」、数式 5 で示された x_1 の状態を表す列の第 1 行は「+1」、よって、 $f_{3,1} \cdot x_{3,1}$ は「 $0 \cdot (+1)$ 」となっている。数式 6 の y_3 行と始点単語列との交点総てについて、その指数と数式 7 に示された対応する状態指数を乗算演算子で結び、付加演算を用いて列記していくと上記の式となる。

ここで、乗算演算子 (\cdot) の演算結果を次のように定義する。

【0 0 6 5】

【数 1 0】

(公理 1)

$$\begin{aligned}
 0 \cdot (-1) &= (+1) & 0 \cdot (\text{null}) &= (+1) & 0 \cdot (+1) &= (+1) \\
 1 \cdot (-1) &= (-1) & 1 \cdot (\text{null}) &= (\text{null}) & 1 \cdot (+1) &= (+1)
 \end{aligned}$$

数式 1 0 の公理は以下のことを示している。始点単語指数が 0 であれば、始点単語の状態が何であれ、始点単語の状態は真 (+1) となり、終点単語の状態に影響を与えない。始点単語指数が 1 の場合は、始点単語の状態はそのまま変化せず、始点単語の状態が終点単語の状態に営業を与える。

【0 0 6 6】

次に、付加演算子 (+) の演算結果を次のように定義する。付加演算子に用いる値は、公理 1 の演算結果である。

【0 0 6 7】

【数 1 1】

(公理 2)

$$\begin{aligned}
 (-1) + (-1) &= (-1) & (-1) + (\text{null}) &= (-1) & (-1) + (+1) &= (-1) \\
 (\text{null}) + (-1) &= (-1) & (\text{null}) + (\text{null}) &= (\text{null}) \\
 (\text{null}) + (+1) &= (\text{null}) \\
 (+1) + (-1) &= (-1) & (+1) + (\text{null}) &= (\text{null}) \\
 (+1) + (+1) &= (+1) \text{ または } (-1)
 \end{aligned}$$

上記の公理 2 は、さらに右辺によって類別してみると、

【0 0 6 8】

【数 1 2】

<第 1 群 : 右辺が (+1) となる場合>

$$(+1) + (+1) = (+1) \text{ [右辺の値はユーザ要件を満たす]}$$

<第 2 群 : 右辺が (-1) となる場合>

$$\begin{aligned}
 (-1) + (-1) &= (-1) & (-1) + \text{null} &= (-1) & (-1) + (1) &= (-1) \\
 \text{null} + (-1) &= (-1) & (1) + (-1) &= (-1) \\
 (+1) + (+1) &= (-1) \text{ [右辺の値はユーザ要件を満たさない]}
 \end{aligned}$$

<第 3 群 : 右辺が (null) となる場合>

$$(\text{null}) + (\text{null}) = (\text{null}) \quad (\text{null}) + (+1) = (\text{null}) \quad (+1) + (\text{null}) = (\text{null})$$

となる。

【0 0 6 9】

数式 1 1 乃至 1 2 から明らかなように、付加演算子(+)の意味については、例えばもし、左辺に (-1) が 1 個でもあれば右辺は (-1) となる。もし左辺に (-1) がなく、かつ空 (null) が 1 個でもあれば、右辺は空である。もし左辺のすべてが (+1) であるなら、右辺の状態は、終点である右辺の値がユーザ要件を満たす場合は (+1)、ユーザ要件を満たさない場合は (-1) である。

。

公理 1 及び 2 を使うことによって、次の定理が証明される。

(定理 1)

終点単語の状態が (+1) (真) になるのは、始点単語の総ての状態が (+1) (真) であり、かつ終点単語の値がユーザ要件を満たす場合である<公理 2 第 1 群>。

(証明)

もし単語の状態が (+1) (真) であるなら、それは公理 1 によって、その単語が始点単語として用いられるとき、 $1 \cdot (+1) = (+1)$ であるから始点単語は (+1) (真)

になる。その単語が始点単語として用いられない場合については、それは $0 \cdot * = (+1)$ ($*$ は 3 つの状態のいずれかを表す) であるから、単語の状態がいずれであっても、公理 2 に用いる始点は $(+1)$ (真) となる。付加演算が実行された結果が $(+1)$ (真) になるのは、公理 2 によって、定理 1 の場合以外には考えられない。

(定理 2)

終点単語の状態が (-1) (偽) になるのは、始点単語の状態が 1 つでも (-1) であるときか、あるいはすべての始点単語が $(+1)$ (真) であってかつ終点単語の値がユーザ要件を満たさない場合である<公理 2 第 2 群>。

(証明)

もしいずれかの始点単語として用いられる単語の状態が (-1) であるなら、公理 1 によって $1 \cdot (-1) = (-1)$ であるから、始点単語の状態は (-1) (偽) になる。そのために、結果が公理 2 によって (-1) (偽) になる。すべての始点単語が $(+1)$ (真) であるときは、定理 1 及び<公理 2 第 2 群>によって、 (-1) (偽) になると証明される。

(定理 3)

終点単語の状態が (未決定である) 空になるのは、1 個以上の始点単語の状態が空であり、かつ始点単語の残りの状態が $(+1)$ (真) である場合である<公理 2 第 3 群>。

(証明)

もし 1 個以上の始点単語の状態が空であって、残りの始点単語の総てが $(+1)$ (真) あるなら、公理 2 によって終点単語の状態は空になる。

一巡処理は $X_m = F X_{m-1}$ によって表せる。繰り返し処理が継続されると以下のようになる。

【0 0 7 0】

【数 1 3】

$$X_1 = F X_0$$

$$X_2 = F X_1 = F^2 X_0$$

⋮

$$X_m = F X_{m-1} = F^m X_0$$

$$X_{m+1} = F X_m = F^{m+1} X_0$$

F^m は通常のマトリックスの積として定義される。 F^m の要素 (i, j) はマイナスでない整数である。

終点単語の状態は一巡処理を繰り返す繰り返し処理によって空から $(+1)$ または (-1) に変化する。

その状態については、たとえ $(+1)$ (真) のものと (-1) (偽) のものとが繰り返し

プロセスを実行するとしても、終点の状態は変化しない。有向グラフによって表現した際、グラフに閉じられたループが存在し、終点が閉じられたループ内にあるとき、あるいは、このシステムが始点を持っていないとき、何度もそれを繰り返すとしても、終点単語の状態は空のままである。「状態に変化がない（すなわち、固定点fixed pointに達した）」ことは $X_{m+1} = X_m$ で表わされ、数式 8 を用いると、次のようになる。

【0 0 7 1】

【数 1 4】

$$F^{m+1} X_0 = F^m X_0$$

この時

F : 隣接行列

X : 状態ベクトル (単語の状態を列で示す)

X_0 : 初期状態、すなわち全ての要素が null

m : 繰り返しの数

この方程式が有効なとき、プログラムは停止する。

I - 3. 不要な繰り返しを回避する方法 (実行順序の並び替え)

I - 3 - (1) 隣接行列の性質

I - 2 で、数学的構造モデルが確立された。ユーザ要件のすべての特性が隣接行列のみで示される。したがって、この隣接行列の特性を調べるのは非常に重要であって、有用である。

【0 0 7 2】

隣接行列 F の特性を利用して、不要な反復を避けるための方法を示す。どんなプログラム (アルゴリズム) も、再帰的な方法のみを用いて説明することができる。手続的には、再帰は while ループ (繰り返しの終了条件を while で指定したループ) によって実現される。

【0 0 7 3】

数式 8 を使用して、「いかなる状態変化もない」(すなわち、「定点に達する」という状態は、以下のように示される:

【0 0 7 4】

【数 1 5】

$$F^{m+1} X_0 = F^m X_0$$

この時

F : 隣接行列

X : 状態ベクトル単語の状態を列で示す

X_0 : 初期の状態。言い換えれば全ての要素が空

m : 繰り返し処理の回数

この公式が有効になるとき、プログラムは止まることになる。

【0 0 7 5】

このとき、行列 F には、どんな特性があるか? 行列 F_m の特性は何であるか? これらの疑問に答えるために、定理 4 を導入することが必要である。

【0 0 7 6】

空でない有限集合 V の組 $G = (V, E)$ と $V \times V$ の部分集合 E は有向グラフと呼ばれる。 V の要素は頂点もしくはポイント (点) と呼ばれ、 E の要素 (u, v) は縁もしくは矢と呼ばれる。 G のポイントの有限系列 P 、 $P = \langle v_1, v_2, \dots, v_n \rangle$ は v_0 から v_1 に至るパスと呼ばれ、各 i ($1 \leq i \leq n$)

)について、 $(v_{i-1}, v_i) \in E$ が成り立つときには、 n はこのパスの長さと呼ばれる。 v_0 は P の始点、 v_n は終点と呼ばれる。 $\langle v_0 \rangle$ は長さゼロのパスである。パスは隣接している矢の方向と共に G のポイントを追従する道筋である。

(定理 4)

有向グラフ V の隣接行列を A と定義する。 A^n の (i, j) 要素は、互いに異なる G の v_i から v_j までの長さ n のパスの数である。

$m=1, m=2, \dots$ と連続して計算し $F^{m+1}X_0 = F^mX_0$ が成立するときにはプログラムが停止する。 F と F^m のプロパティを検証する。 X_0 の全要素が常にヌル(空)なので、 F はシステムの全プロパティを表す。

1) $F^m=0$ である場合 ($m \leq n$)

F^m のすべての要素が0である。定理 4 を用いれば、閉鎖ループがこの隣接行列の有向グラフに存在しないことが証明される。変数 m はその有向グラフの最も長いパスの長さを表す。 F の固有値はすべて0である。零べき行列($F^m=0$)のプロパティから、対角に要素0を持つ下三角形のマトリクス F に再配列することができる。即ち、

$F' = P^{-1}FP$ 、ここで P は正方行列もしくは長方形行列。

理由は、 $F(F^mX_0) = F^mX_0, F^{m+1}X_0 = F^mX_0$

が有効であるからである。 X_m のすべての要素は(+1)か(-1)である。

2) $F^m \neq 0$ である場合 ($m \leq n$)

もし、閉じたループがこの隣接行列の有向グラフに存在した場合、対角の要素

【0 0 7 7】

【数 1 6】

$$f_{ii}^{(r)} : i = 1, \dots, n \text{ of } F^r$$

に1つ以上の整数が現れることが、定理 4 を使って証明される。このとき、変数 r は最も短い閉じたループである。 F の固有値は、総てが0ではない。言い換えれば、固有値の1つは0と等しくない。総ての X_0 値が空(null)なので、 $F^{m+1}X_0 = F^mX_0$ は有効である。変数 m は、その有向グラフのループの前の最も長いパスの長さを表す。 X_m の例では、

【0 0 7 8】

【数 1 7】

$$X_m = '[(+1), (+1), (\text{null}), (\text{null}), (\text{null}), (\text{null})]。$$

もし、閉じたループが有向グラフに存在しなければ、その隣接行列は零冪行列($F^m=0$)となる。この特性は重要である。

I - 3 - (2) 繰り返しプロセスを不必要にする単語の並べ方

「有向グラフ」で表現する際に、前述の第III章1項で示したように、閉じられたループ内で始点単語・終点単語である関係がないときには、隣接行列が下三角行列になり、対角の三角形が0になるような一巡処理順序に単語を並べることによって、繰り返しプロセスを避けることができる。

【0 0 7 9】

分析的に規則正しい行列 P を得るのは困難である。 $F' = P^{-1}FP$ を用いる代わりに、トポロジカル・ソートを用いる。半順序関係が与えられているとき、最初にするべき仕事から

終わりにするべき仕事まで、データを一行に並べることをトポロジカル・ソート (topological sort) と呼ぶ。半順序関係 (partial order) であるから、答えが常に唯一的に決定されるわけではない。

【0080】

有向グラフに深さ優先探索 (depth first search: 縦型検索ともいう) をしつつ、戻るときにノードを拾い上げるという方法によって、トポロジカル・ソートを実行することができる。各ノードからこれを行い結果を逆順にならべる。順序処理 (一巡処理) における単語を並べるには、この方法によって単語を並べさえすればよい。

図4は、あるノード1乃至8が形成する特定の順序関係を表した有向グラフである。

【0081】

即ち、同図に示す例においては、先に定義した始点単語と終点単語との関係を用いて次のように規定できる。

ノード2 ≤ ノード1

ノード1 ≤ ノード3

ノード3 ≤ ノード4、ノード5 ≤ ノード4

ノード2 ≤ ノード5

ノード5 ≤ ノード6

ノード3 ≤ ノード7、ノード6 ≤ ノード7、ノード8 ≤ ノード7

ノード6 ≤ ノード8

(例4)

図4に示される有向グラフの隣接行列は次の数式18の通りである。

【0082】

【数18】

$$F = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{matrix}, X = \begin{bmatrix} \text{null} \\ 1 \\ \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \end{bmatrix}$$

図4の有向グラフでは閉じられたループ内で始点単語・終点単語の関係がない。そこで、この半順序関係に係る隣接行列が下三角行列になるように、トポロジカル・ソートを行う。具体的には次のように行う。

有向グラフに対する深さ優先探索を実行する。即ち、一例として、ノード1を始点とすると、ノード1を出発して連結されている未訪問のノードを半順序関係を守って探索 (即ち

、連結される未訪問のノードまで進み、行き場所がなくなったら、行ける場所があるノードまで戻って（再帰）再び進めるだけ進み、行き場所がなくなったら終了（再帰からリターン）：これを「再帰的に進む」ともいう）する。これにより、ノード1→ノード3→ノード4（*）→ノード3→ノード7（*）→ノード3（*）→ノード1（*）を得る。但し、（*）は再帰の時点を示す。

次に、始点をノード2に移し、上記と同様の探索を行う。すでに訪問したところは進まない。これにより、ノード2→ノード5→ノード6→ノード8（*）→ノード6（*）→ノード5（*）→ノード2（*）を得る。

その他のノードを始点として探索をするところ、すでに全ノードを訪問済みなので探索は終了する。

戻る（再帰の）ときのノードを拾い上げる。具体的には、（1）（2）で（*）の時点のノードを拾い上げる。上の例では、ノード4（*）→ノード7（*）→ノード3（*）→ノード1（*）→ノード8（*）→ノード6（*）→ノード5（*）→ノード2（*）を得る。

順序処理（一巡処理）における単語を逆順に並べる。即ち、（4）で拾い上げたノードの列を逆に並べたものを一つの解として得る。上の例では、ノード2→ノード5→ノード6→ノード8→ノード1→ノード3→ノード7→ノード4が求める一つの解である。

以上より、トポロジカル・ソートの結果は（1, 2, 3, 4, 5, 6, 7, 8）→（2, 5, 6, 8, 1, 3, 7, 4）である。（「ノード」の表記は省略する。）

このトポロジカル・ソートの具体的な方法についてフローチャートを用いて説明すると次のようになる。図5はトポロジカル・ソートの動作を説明するためのフローチャートである。

【0083】

まず、初期値を、 $a=1, I=1$ と設定する（ステップ501）。

【0084】

次に、 $a=a+1$ とし、始点をノード a に設定する（ステップ502）。

【0085】

ここから探索を開始する。具体的には、訪問可能な連結されているノードで未訪問のところがないかを判定する（ステップ503）。これについては、例えば、各ノードに訪問フラグを設定、初期値に0（未訪問を示す）を入れておき、当該ノードが訪問されたときには訪問フラグを1にするなどの方法により自動判定させることができる。

【0086】

ここで未訪問ノードが存在するときには、1ノード進み（ステップ504）、ステップ503に戻る。

【0087】

未訪問ノードが存在しないときには、1ノード戻り（再帰）、 $I = I + 1$ とする（ステップ505）。このとき、再帰に係るノードを再帰ノード列 I 番目の要素として退避する（ステップ506）。

【0088】

次に、再帰の結果ノード a に戻ったかを判定し（ステップ507）、まだ戻っていない場合はステップ503に戻る。ノード a に戻った場合には始点をノード a とする探索は終了する。即ち、再帰からリターンし、 $I = I + 1$ とし（ステップ508）、再帰点を先の再帰ノード列 I 番目の要素として退避する（ステップ509）。

【0089】

ここで全ノードが探索済みかどうかをチェックする（ステップ510）。まだ探索を行っていないノードが存在する場合には、始点を替えて再度探索を始めるため、ステップ502に戻る。全ノードの探索が終了しているときには、探索は終了し、退避してあった再帰ノード列を逆順に並べる（ステップ511）ことで、並び替えの1候補を得て、一連の

トポロジカル・ソートの動作を終了する（ステップ 5 1 2）。

【0 0 9 0】

このように、トポロジカル・ソートのアルゴリズム自体を上記で説明したフローチャートに沿ってプログラム化すれば、トポロジカル・ソートの自動プログラムを得ることができる。

【0 0 9 1】

さて、上記により F をトポロジカル・ソートした。その結果、F'（トポロジカル・ソートされた F）は次の数式 1 9 のようになる。

【0 0 9 2】

【数 1 9】

$$F' = \begin{matrix} & \begin{matrix} 2 & 5 & 6 & 8 & 1 & 3 & 7 & 4 \end{matrix} \\ \begin{matrix} 2 \\ 5 \\ 6 \\ 8 \\ 1 \\ 3 \\ 7 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{matrix}$$

トポロジカル・ソートの結果、一巡処理の順序は、「1, 2, 3, 4, 5, 6, 7, 8」から「2, 5, 6, 8, 1, 3, 7, 4」となった。

【0 0 9 3】

このように F' が得られるが、この F' はその形から明らかなように上三角形に当たる総ての項が 0 である下三角行列になっている。この（ソート済み隣接行列である）F' は（上三角形部分には 0 しかないため）その構造上、あるノード（終点単語）を求めるための始点単語は当該ノードより（ソート済み隣接行列における）上の行のノードにしかない。従って、F' の行の上位から順番に値を求めていけば、F で同じことを行う場合に比べて、繰り返しや遡りを行うことなくスムーズに総てのノードについて値が求まる。

【0 0 9 4】

さて、次に、上記で説明したトポロジカル・ソートを L y e e（登録商標）方法論において応用する方法の一つを説明する。

【0 0 9 5】

図 6 は、L y e e（登録商標）方法論における単語単位の論理要素をノードとして上記のトポロジカル・ソートを用いて繰り返しをなくすための自動制御の動作を説明するため

のフローチャートである。

【0096】

同図に示すように、まず、単語ごとの論理要素（これらをノードと考える）間の関係を規定する（ステップ601）。

【0097】

このためには例えば有向グラフの作成（ステップ602）を行っても良いが、これに限定されるものではない。

【0098】

次に、ノードについての（上記説明した）隣接行列を定義する（ステップ603）。

【0099】

この隣接行列に対して上述のトポロジカル・ソートを実行する（ステップ604）。トポロジカル・ソートの詳しいアルゴリズムについては図5及び対応する説明で詳述したので、ここでは割愛する。また、ここでは探索技法の一例としてトポロジカル・ソートとして説明したが、このステップ604では前述の他の探索の実行を行っても良いのはもとよりである。

【0100】

最後に、トポロジカル・ソートの結果得られた1解に基づき、当該単語単位の論理要素（ノード）を並び替え（ステップ605）、当該動作を終了する（ステップ606）。

【0101】

なお、隣接行列定義のステップ（ステップ603）については、本稿では理解の容易さを助けるために入れたが、本質的には必須のステップではない。つまり、隣接行列を定義することなく、単語ごとの論理要素間関係の規定（ステップ601）から、直接に、或いは有向グラフ作成（ステップ602）を介して、トポロジカル・ソート等の探索のステップ（ステップ604）に至ることも本発明の思想の範囲内であり、本発明の目的とするところを実現し得る。

【0102】

このように、L y e e（登録商標）のソフトウェア生成自動プログラムにかける前の段階で、L y e e（登録商標）方法論によるユーザ要件から抽出した単語ごとの論理要素に対してトポロジカル・ソートをかけることで繰返しの省かれたL y e e（登録商標）方法論によるソフトウェア生成が可能となる。

【0103】

換言すれば、上記のノードをL y e e（登録商標）方法論における単語ごとの論理要素と考えれば、上記の方法でF'を求めることで、ある単語について値を求める（有意性を決定する）に当たって、何回も繰返し処理を行うことなく1回のRUNのみで目的を達することが可能となる。

【0104】

さらに、上記のようなトポロジカル・ソートのアルゴリズムは、上記の説明から明らかのように、それ自体をソフトウェアとしてプログラミングして自動化することが可能である。

【0105】

従って、L y e e（登録商標）方法論によるソフトウェア自動生成方法についてそれをさらに効率化・高品質化するのにも自動化を図れることになる。

【0106】

なお、上記の説明においては、有向グラフに表される単語の順序関係として主として半順序関係を持つ場合について説明したが、半順序関係のみならず、全順序関係について、或いは全順序関係及び半順序関係が混在する場合についても上記の説明が該当することはもとよりである。

【0107】

また上記の説明においては、理解のしやすさの観点から、有向グラフの作成を行った後に隣接行列を作成する点を説明したが、有向グラフの作成を介さずに、単語単位のプログ

ラム相互の関係から直接隣接行列を作成するようにしても本発明は実現できる。

すべての単語で作るべきプログラムの構造と当該プログラムを制御するのに用いるパレット連鎖関数のプログラムの構造は、上記の記述で明確になった。そして、それらが数学上適切に動くことが証明された。

I-4. まとめ

以上詳細に説明したように、トポロジカルソート技術では、単語の状態を定義している単語単位をベースにするプログラムの数学的な構造モデルを作成し、それをL y e e（登録商標）方法論によるソフトウェア開発に適用することで、より高効率、高速、高パフォーマンスの開発を実現した。

【0108】

単語の状態は完全な半順序を構成する。それは単語の最小要素（極小点）として 空を持っている。有向グラフとその隣接行列とを使って、この構造モデルは数学的な面からも正しいといえる。

【0109】

さらに、本技術ではパレットの一般的な考えと基本構造の正当性を考察した。それらは実務上、プログラムを作るために必要である。また、順次読込 (sequential read) のようなアルゴリズム、合計値算出を実現するための構造について説明した。これらの一般的な考えは Lyee（登録商標）のシステム開発方法論で使われる。それは多くの産業的な応用分野ですでに実用的に有効とされた多くの結果をもたらしている。

人間であるユーザと、外の装置とのインタフェース及びコミュニケーションについても論じられている。これは、このようなシステムのタイプのために、単語単位をベースとするプログラムとして設計する（モデリング）ことの、実際の作業を正当化する。

以上を要するに、本技術によれば、数学的な考え方、手法、解析法をL y e e（登録商標）のソフトウェア開発論、開発技術に適用・応用するので、ソフトウェア開発をより効率良く、高速に、精密に実現することができる。

【0110】

上記の詳細な説明では、有向グラフ表記、隣接行列算出、トポロジカル・ソート、並び替えという順序で本発明の思想を実現する方法について述べた。しかしこれらはどれも絶対不可欠な要素というものではなく、例えば、有向グラフ表記の表記を経ないで直接隣接行列算出→トポロジカル・ソート→並び替えという順序で本発明の思想を実現してもよい。

【0111】

さらに、有向グラフに対して、例えば幅優先探索 (Breadth-First Search)、反復深化法 (Iterative Deeping)、発見的探索 (Heuristic Search)、ヒル・クライミング (Hill Climbing)、最適優先探索 (Best-First Search)、オイラー (Euler) の一筆書き、ダイクストラ (Dijkstra) 法等の各種探索技法を組み合わせることで並び替えてもよい。

【0112】

或いは、隣接行列算出のあとを、これらの各種探索技法を組み合わせることで本技術の根本思想を実現してもよい。

【0113】

図7は、本技術をソフトウェア開発装置として実現する場合の機能構成を示したブロック図である。即ち、同図に示す一例においては、ソフトウェア開発装置1000として、

次の要素を備えさせる。

【0114】

制御部1010は全体の制御及びL y e e（登録商標）方法論に基づいたソフトウェア開発を自動的に行う機能を有する。即ち、後述する（有向グラフ作成部1050、）隣接行列計算部1060及びトポロジカルソート部1070により並び替えられた単語単位プログラム群をL y e e方法論に自動的に適用して生成コードを得る機能を有している。

【0115】

情報入力部1020は（図示しない）画面等を通じて、L y e e（登録商標）に係るユーザ要件を初めとする人間の意思をコンピュータに入力するインターフェースとしての機能を果たす。情報取得部／通信部1030は外部との通信や（ソフトウェア開発装置1000にある場合の）データベース1040、1041等との情報の授受を行う。なお、この例ではデータベース1040、1041はソフトウェア開発装置1000内に持っていてよいことはもとよりである。

【0116】

有向グラフ作成部1050は、制御部1010の制御に基づき、上記で説明したように、L y e e（登録商標）に係るユーザ要件から半順序関係を割り出し、それを有向グラフとして作成する機能を有する。隣接行列計算部1060は、制御部1010の制御に基づき、有向グラフ作成部1050で作成された有向グラフについて、上記で説明したような隣接行列を計算・作成する。トポロジカルソート部1070は、制御部1010の制御に基づき、隣接行列計算部1060で計算・作成された隣接行列に対して、上記で説明したようなトポロジカルソートを実行する機能を有する。

【0117】

なお、上記トポロジカルソート部1070は探索を行う機能を持つ一例であって、上述のように、他の探索手法を実現する機能を有する探索部（一般）としても本発明の思想は実現される。

【0118】

このように構成することにより、これら各部（1010～1070、従って1000）は総て自動化できるので、結局繰返し処理の排除→ソフトウェア生成まで一貫して自動処理が可能となり、ソフトウェア生産速度、効率は各段に向上する。

【0119】

このように、各機能を、それを実現するようにプログラムとして組んだソフトウェアとして、或いはそのようなソフトウェアを例えばROM（Read Only Memory）に書きこみかかるROMを組み込んだ全体システム（装置）とすることで、前述した本発明に係る各要素をソフトウェアとして或いはそのようなソフトウェアが組み込まれたソフトウェア開発装置としても実現可能である。

II章 L y e e方法論のシステム構造の単一化

本章では、I章で述べた単語単位のプログラムの最適順序への並び替えによるプログラムの効率化技術を、システム全体に適用してシステム全体の効率化を実現するための、L y e e方法論のシステム構造の単一化技術について述べる。

II-1 L y e e方法論のシステムの構造

II-1-1 L y e e方法論のシステムを構成する単位：基本構造

L y e e方法論は、同一条件を持っている単語を1つのグループとして、システム全体をそのグループ単位に分割する。同一条件とは、同時に出力（画面またはファイルへ）される、という条件で、そのグループの単位を論理レコードと呼ぶ。論理レコード単位に単語単位のプログラムが基本構造（Basic Structure）と呼ぶ構造を作り、基本構造単位に単語単位のプログラムが実行される。これは L y e eの規則の1つである。

【0120】

従って、L y e e方法論においては、1つのシステムはいくつかの基本構造に分割され

ることになる。システムのこの構造は、処理経路図 (Process Route Diagram。あるいはPRDと呼ぶ) によって表現される。処理経路図では、基本構造のリンクの状態 (実行の経路を示す) が表現されている。図 8 は 2 つの基本構図からなるシステムの場合の処理経路図の例である。

【0 1 2 1】

基本構造から別の基本構造へとプログラムの実行が遷移するしくみは、図 8 の例で説明すると次のとおりである。第 1 の基本構造において、1 つ以上の始点単語 (start-word) の状態が null (未決定) であることによって終点の値の状態が null (未決定) である出力単語が 1 つ以上あるとき、次に実行されるべき基本構造 (第 2 の基本構造) が深さ優先探索 (depth-first search) によって探索され、次に実行する基本構造として指定される。次に実行される第 2 の基本構造において、値が null である出力単語がなくなったとき、第 2 の基本構造は実行を終了し、次に実行する基本構造として、null 値が最初に現われた第 1 の基本構造が指定される。このプロセスは、システム全体の単語の状態変化が総て消滅するまで継続される。

このように、L y e e 方法論によるシステムは、基本構造単位の構造を持つので、システムの実行は基本構造単位の実行順序となる。この場合、基本構造単位の実行は 1 巡処理で完了することは保証されていおらず、単語単位のプログラムの実行は無駄な繰り返しが発生する。

【0 1 2 2】

これに対して、システムとして実行時の構造として、システム全体を 1 つの構造に統合すれば、基本構造単位での順序ではなく、システム内の単語単位プログラム全部に対して、一括に I 章で述べたトポロジカルソートを施すことができ、単語単位プログラムを最適順序で実行することによってプログラムの実行時間等の効率化がはかれる。

基本構造から次に実行すべき基本構造の決定が、深さ優先探索では決定できない場合がある。それは、ユーザの画面からの指示によって実行される基本構造が決定する場合である。たとえば、画面に処理 A を実行するボタンと、処理 B を実行するボタンがあり、ユーザがこれらのボタンのいずれか 1 つを押下することによって、プログラムが実行する処理内容が決定するような場合である。このように、画面からのユーザの指示によって処理内容が決定するような仕様を含むシステムにおいては、基本構造単位に分割され単語単位プログラムの実行条件をそのままにして、基本構造を 1 つの構造に統合し、トポロジカルソートを施すことができない。以下に、画面からのユーザ指示が実行する基本構造を決定するような場合の基本構造の単一化の方法について具体例を上げて以下に詳しく述べる。

そのためのステップとして、まず、基本構造をリンクする (次に実行すべき基本構造を指定する) 役割を果たしているプログラム、経路作用要素について詳しく考察する。

I - 1 - (2) 基本構造をリンクするプログラム：経路作用要素

次に実行する基本構造を指定することによって基本構造をリンクする役割は、経路作用要素 (Routing Vector) と呼ぶプログラムが行う。L y e e の単語単位プログラムにおいては、入力単語、状態単語の値を生成するための単語のプログラム以外に、次に実行する基本構造を指定する、あるいは出力単語の値を出力する、などの作用を行うためのプログラムがある。これらのプログラムを作用要素 (action vector) と呼び、経路作用要素もその 1 つである。

これらの作用要素も一種の単語と見なすことができ、実際これらのプログラムの基本的構造は単語のプログラムと同じである。単語と作用要素の大きな違いは、単語は 1 つの単

語のみに作用（値の生成という作用）するが、作用要素は同時に 2 つ以上の単語に対して作用する点だけである。たとえば、経路作用要素（次に実行する基本構造を指定する）は、基本構造という単語の集合に対して作用し、出力作用要素（出力単語を出力する）は、出力レコードという単語の集合に対して作用する。作用要素を単語とみなすことによって、作用要素を有向グラフおよび隣接行列の要素として取扱うことができ、システム全体を I 章で説明してきた数学的モデルによって表すことができる。

【0 1 2 3】

以下に例を上げて、単語単位プログラムのモデルにおける経路作用要素の意味を説明する。

（例 5）

図 9 は、あるシステムの画面で、ユーザがデータ項目 c およびデータ項目 d にデータを入力して実行ボタンの a または b を押すことによって、データ項目 g に値を得るためのシステムである。7 0 1 はデータ項目 c の入力データフィールド、7 0 2 はデータ項目 d の入力データフィールド、7 0 3 はデータ項目 g の出力データフィールド、7 0 4 は実行ボタン a、7 0 5 は実行ボタン b である。実行ボタン a が押されるときは、g の値は式 $c + d$ で計算され、実行ボタン b が押されるときは、g の値は式 $c \times d$ によって計算される。実行ボタンの a および b は、両方が同時に押されることはないので、g の値の生成のための 2 つの計算式のどちらが実行されるかの条件となっている。

【0 1 2 4】

このシステムを基本構造単位に単語で表すと図 1 0 のようになる。図 1 0 の B S 1、B S 2、B S 3 はそれぞれ基本構造である。基本構造 B S 1 は、図 9 に示した画面の基本構造である。B S 1 は、実行ボタンである入力単語 a と入力単語 b、入力データ項目である入力単語 b と入力単語 c、出力データ項目である出力単語 g、経路作用要素である X_r を含んでいる。

【0 1 2 5】

基本構造 B S 2 は、実行ボタン a が押された時、出力データ項目 g の値を式 $c + d$ によって計算するための基本構造（媒体がファイル）である。式 $c + d$ の結果を記録する単語 e を含んでいる。基本構造 B S 3 は実行ボタン b が押された時、出力データ項目 g の値を式 $c \times d$ によって計算するための基本構造（媒体がファイル）である。式 $c \times d$ の計算結果を記録する単語 f を含んでいる。

【0 1 2 6】

画面にユーザがデータを入力した（B S 1 が実行された）後に、出力単語 g の値を計算するための基本構造は B S 2 または B S 3 であるが、どちらが実行されるべきかを指定するのが経路作用要素 X_r である。2 つの基本構造に所属する単語は、それぞれボタン a と b のどちらが押されたかによって実行が決まるので、経路作用要素 X_r は、実行ボタン a が押された場合（単語 a = 真、値がある）は B S 2 を、実行ボタン b が押された場合（単語 b = 真、値がある）は B S 3 を、次に実行する基本構造として指定する。また、実行が次の基本構造に移るのは入力データがそろったときであるので、単語 c および単語 d に値があることも、経路作用要素 X_r の実行条件となる。

【0 1 2 7】

基本構造 B S 2 または B S 3 の実行によって、出力単語 g の値が単語 e または単語 f に成立する。出力データを画面に出力するための基本構造 B S 1 の出力単語 g は、単語 e に値が成立したとき（e = 真）は $g = e$ 、単語 f に値が成立したとき（f = 真）は $g = f$ 、と定義される。

【0 1 2 8】

上記のような単語の定義を整理すると下記の表 2 のとおりである。

【0 1 2 9】

【表 2】

単語	入出力属性	定義式	定義式実行条件
a	入力	—	—
b	入力	—	—
c	入力	—	—
d	入力	—	—
x_r	(経路作用要素)	BS 2 を指定	(1) $a = \text{真}$ (ボタン a が押され、a に値がある) かつ、 $c = \text{真}$ および $d = \text{真}$
		BS 3 を指定	(2) $b = \text{真}$ (ボタン b が押され、b に値がある) かつ、 $c = \text{真}$ および $d = \text{真}$
e	出力	$c + d$	x_r が BS 2 を指定
f	出力	$d \times d$	x_r が BS 3 を指定
g	出力	e	$e = \text{真}$ (e に値がある)
		f	$f = \text{真}$ (f に値がある)

II-2 基本構造の統合によるシステム構造の単一化

II-2-(1) 基本構造の隣接行列

図 11 は、図 10 の単語の関係を有向グラフによって示した図である。基本構造 BS 1 に属する入力単語である単語 a、単語 b、単語 c、単語 d は、有向グラフによって表される単語のネットワークの最端始点となる。同じく基本構造 BS 1 に属する経路作用要素である単語 x_r は、上記表 2 に示したとおり、値が成立（次に実行する基本構造決定の真偽判定が成立）するためには、単語 a または単語 b と単語 c と単語 d が必要であるので、これらの 4 つの単語を始点とする矢印によって 4 つの単語とリンクされる。基本構造 BS 2 に属する単語 e、および、基本構造 BS 3 に属する単語 f は、値が成立するために、単語 c、単語 d および単語 x_r が必要であるので、これらの 3 つの単語を始点とする矢印によって各々 3 つの単語とリンクされる。基本構造 BS 1 に属する出力単語 g は、値が成立するために、単語 e または中間単語 f が必要であるので、これらの 2 つの単語を始点とする矢印によって 2 つの単語とリンクされる。

【0130】

次に、図 11 で有向グラフで示した例 5 のシステムを隣接行列を用いて表す。最初に図 11 の、入力単語を扱った基本構造 BS 1 (901) を表した隣接行列 F1 を下記の数式 20 に示す。

【0131】

基本構造 BS 1 (入力単語) の隣接行列

【数 20】

$$F1 = \begin{matrix} & \begin{matrix} a & b & c & d & x_r \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ x_r \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

BS 1 (入力単語) の要素は単語 a、単語 b、単語 c、単語 d、単語 x_r であるから、F 1 の行列はこれらの要素によって構成される。入力単語である単語 a、単語 b、単語 c、単語 d は、始点単語を持たないので、終点単語 a、b、c、d のそれぞれの横列 (始点単語 a、b、c、d、 x_r との交点) は総て始点単語として用いないことを示す「0」となる。経路作用要素である単語 x_r は、図 11 の有向グラフに示したとおり、始点として単語 a、単語 b、単語 c、単語 d を持つので、終点単語 x_r の横列は、これらの始点単語との交点は始点であることを示す「1」となる。

【0132】

次に、基本構造 BS 2 および BS 3 (以下、基本構造 BS 2 & BS 3 と記す) を合わせたものを表す隣接行列 F 2 を下記の数式 2 1 に示す。

【0133】

基本構造 BS 2 & BS 3 の隣接行列

【数 2 1】

$$F2 = \begin{matrix} & e & f \\ \begin{matrix} e \\ f \end{matrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{matrix}$$

BS 2 & BS 3 の要素は単語 e と単語 f のみであるので、F 2 の行列はこれら 2 つの要素で構成される。単語 e と単語 f はお互いに独立している (すなわち、どちらの単語も始点単語としても一方の単語を用いていない) ので、いずれの交点も「0」となる。

【0134】

基本構造 BS 1 の出力単語を扱った集合の隣接行列 F 3 を、下記の数式 2 2 に示す。

【0135】

基本構造 BS 1 (出力単語) の隣接行列

【数 2 2】

$$F3 = \begin{matrix} & g \\ g & [0] \end{matrix}$$

BS 3 の要素は単語 g のみであるので、F 3 の行列は単語 g 1 つのみで構成される。単語 g は始点として自身を用いないので、交点は「0」となる。

【0136】

以上で、例 5 のシステムを構成する基本構造単位に、同一基本構造内の要素間の関係を隣接行列に表した。

II-2-(2) 結合隣接行列による基本構造の統合

次に、これらをシステム全体として 1 つの隣接行列として表すために、各基本構造の要素である単語と他の基本構造の要素である単語との関係を示すことによって基本構造をリンクする結合隣接行列 (connection matrix) について説明する。

任意の基本構造の単語と他の基本構造の単語の関係は、図 11 に示したように終点単語と始点単語の関係によって表わされている。基本構造同士は、それに属する単語の関係によってリンクされていると言える。図 11 から以下のことが分かる。

- 1) 基本構造 BS 1 (入力) の単語を終点単語としたとき、いずれの他の基本構造の単語も、BS 1 (入力) の単語より後に現れるので、始点単語となりえない。
- 2) 基本構造 BS 2 & 3 の単語を終点単語としたとき、BS 2 & 3 の単語より前に現れる BS 1 (入力) の単語は始点単語となりえるが、BS 1 (出力) の単語は始点単語となりえない。

3) 基本構造BS1 (出力) の単語を終点単語したとき、BS2 & 3およびBS1 (入力) のいずれの単語も、BS1 (出力) の単語より前に現れるので、始点単語となりえる。

上記の考察の結果、基本構造をリンクするための結合隣接行列として重要なのは、任意の基本構造と、それに属す単語の始点単語になりえる単語を含む基本構造との関係を表す結合隣接行列であることがわかる。なぜなら、有向グラフで示されるように、単語間の関係は始点単語から終点単語へと向かう関係のみが存在するのであるから、基本構造間の関係も、それと同様に始点単語を含む基本構造から終点単語を含む基本構造へと向かう関係のみが存在し、逆の関係は存在しない(そのような基本構造間の関係を表す結合隣接行列は、すべての交点が0となる)からである。

【0137】

以下に、例5の基本構造をリンクする(始点単語を含む基本構造から終点単語を含む基本構造へと向かう関係を表す)結合隣接行列を1つずつ考察する。

数式23は、基本構造BS1 (入力) (始点単語) から基本構造BS2 & BS3 (終点単語) への関係を示す結合隣接行列FC1である。

【0138】

【数23】

$$FC1 = \begin{matrix} & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} & \mathbf{x}_r \\ \mathbf{e} & \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \end{bmatrix} \\ \mathbf{f} & \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \end{bmatrix} \end{matrix}$$

隣接行列FC1は、基本構造BS2 & BS3の要素である単語eと単語fを終点単語においたとき、BS1 (入力) の要素である単語a、単語b、単語c、単語d、単語 x_r が始点単語として用いられる状態を表し、2つの基本構造の関係を示している。図11で示したように、単語eおよび単語fは、単語c、単語d、単語 x_r を始点単語に用いる。従って、FC1において、終点単語eおよび終点単語fの行と、始点単語c、d、 x_r との交点は「1」となり、その他の始点単語との交点は「0」となる。

【0139】

数式24は、基本構造BS1 (入力) および基本構造BS2 & BS3 (始点単語) から、基本構造BS1 (出力) (終点単語) への関係を示す隣接行列FC2を表している。

【0140】

【数24】

$$FC2 = \mathbf{g} \begin{matrix} \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} & \mathbf{x}_r & \mathbf{e} & \mathbf{f} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{matrix}$$

隣接行列FC2は、基本構造BS1 (出力) の要素である単語gを終点単語においたとき、有向グラフにそれ以前に現われるBS1 (入力)、BS2、BS3の要素である単語a、単語b、単語c、単語d、単語 x_r 、単語e、単語f、が始点単語として用いられる状態を表して、2つのグループの関係を表している。図11で示したように、単語gは、単語eと単語fを始点単語に用いる。従って、FC2において、終点単語gの行と、始点単語eおよびfとの交点は「1」となり、その他の始点単語との交点は「0」となる。

【0141】

以上が、例5のシステムを構成する基本構造間のリンクに意味を持つ結合隣接行列であった。システム全体として1つの隣接行列へ統合する過程の説明を簡便にするために、下記に、関係が存在しない基本構造間の結合隣接行列も示す。数式25は、基本構造BS2 & BS3 (始点単語) から基本構造BS1 (入力) (終点単語) の関係がないことを示す

結合隣接行列 $FC3$ である。

【0 1 4 2】

【数 2 5】

$$FC3 = \begin{array}{c} \begin{array}{cc} & \begin{array}{cc} e & f \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ x_r \end{array} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \end{array}$$

数式 2 6 は、基本構造 $BS1$ (出力) (始点単語) から基本構造 $BS1$ (入力) (終点単語) の関係がないことを示す結合隣接行列 $FC4$ である。

【0 1 4 3】

【数 2 6】

$$FC4 = \begin{array}{c} \begin{array}{c} g \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ x_r \end{array} \end{array}$$

数式 2 7 は、基本構造 $BS1$ (出力) (始点単語) から基本構造 $BS2 \& BS3$ (終点単語) の関係がないことを示す結合隣接行列 $FC5$ である。

【0 1 4 4】

【数 2 7】

$$FC5 = \begin{array}{c} \begin{array}{c} g \\ \begin{array}{c} e \\ f \end{array} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ x_r \end{array} \end{array}$$

上記に示した隣接行列 $F1$ から $F3$ 、結合隣接行列 $FC1$ から $FC5$ を用いて、例 5 のシステムを 1 つの隣接行列 F として表したものが数式 2 8 である。

【0 1 4 5】

【数 2 8】

$$F = \begin{bmatrix} F1 & 0 & 0 \\ FC1 & F2 & 0 \\ FC2 & & F3 \end{bmatrix} = \begin{matrix} & \begin{matrix} a & b & c & d & x_r & e & f & g \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ x_r \\ e \\ f \\ g \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

上記数式で、「F=」の右辺に示された、F 1、F 2、F 3、F C 1、F C 2と 0（行列の交点が総て 0 である隣接行列で、F C 3、F C 4、F C 5 に相当する）で構成された行列は、基本構造単位の隣接行列と結合隣接行列がどのように統合されてシステム全体を表わすかを示している。次に、それらの要素を単語に変えたものが、その右辺にある a、b、c、d、x_r、e、f、g を行列の要素とする隣接行列である。F 1、F 2、F 3、F C 1、F C 2、F C 3、F C 4、F C 5 がどのように統合されて F を構成するのか、さらにわかりやすく示したのが図 1 2 である。図 1 2 の 1 0 0 1 の部分が F 1、1 0 0 2 が F 2、1 0 0 3 が F 3、1 0 0 4 が F C 1、1 0 0 5 が F C 2 である。行列の交点が総て 0 である結合隣接行列は、F C 3 が 1 0 0 6、F C 4 が 1 0 0 7、F C 5 が 1 0 0 8 を構成する。

【0 1 4 6】

以上のようにして、2 つ以上の基本構造から成る 1 つのシステムは、1 つの隣接行列で表すことができる。

以上でシステム全体を 1 つの隣接行列で表わすことができたので、このシステムの隣接行列に対して、I 章で述べたトポロジカルソートを行って、総ての単語の値の状態が最少実行回数で確定（すなわち総ての出力単語の生成が完了）するような最適順序に単語単位のプログラムの並び替えを行う。

【0 1 4 7】

例 5 のシステムの隣接行列 F をみれば、図 1 3 に示すように、値が 1 である始点単語との交点は総て左下三角形（1 3 0 1）の内にある。I 章で述べたように、これは単語が最適順序に並んでいることを意味する。従って、数式 2 8 の隣接行列 F は、すでにトポロジカルソート済みであって、初期値の単語の状態ベクトルを与えれば、最少実行回数で出力単語の生成を完了することができる単語の順序になっているといえる。

ここで、トポロジカルソートを施す効果について補足する。トポロジカルソートによって最適実行順序に並んだ単語単位のプログラムは、1 巡処理で総ての出力単語の値を生成することができる。しかし、システム全体に対して言及する場合には、より正確には「最少実行回数」で実現できるということが適切である。なぜなら、システム全体でみたときには、集計処理のように、一時的記録領域を置くことによって同じ計算式を繰り返し利用する必要がある処理を含む場合があるからである。もちろん、この場合の「繰り返し」は、無駄な繰り返しではない。集計するデータの数マイナス 1 が最少繰り返し数になる。こ

のような集計処理も、結合隣接行列を用いて 1 つの隣接行列に表わすことができる。

I I - 2 - (3) システムの隣接行列の検証

本セクションでは、前セクションで示したシステムの隣接行列 F が、出力単語の値を生成する関数として機能し、しかも、トポロジカルソート済みであるので、最少実行回数で総ての出力単語の生成を完了できることを、I 章で述べた単語の状態ベクトルをかける計算操作によって検証する。出力単語 g の値が決定されれば、隣接行列 F は関数として機能すると言える。また、例 5 のシステムは、前述の集計処理のような繰り返し処理を含まないので、1 巡処理で出力単語 g の値の状態が確定すれば、最少実行回数で処理が完了したと言える。

例 5 のシステムにおいて、ユーザが入力を行う前の単語の値の状態、すなわち単語の状態ベクトル X の初期値は、I 章で述べたように総ての要素が「null (未決定)」で、下記の数式 2 9 のようになる。

$$X = \begin{bmatrix} a & \text{null} \\ b & \text{null} \\ c & \text{null} \\ d & \text{null} \\ x_r & \text{null} \\ e & \text{null} \\ f & \text{null} \\ g & \text{null} \end{bmatrix}$$

数式 2 8 の隣接行列 F (すなわち例 5 のシステム) に、数式 2 9 の状態ベクトル X をかける計算処理を施す (すなわち、例 5 のシステムに入力を行う)。 $F X$ の計算処理によって変化する単語の値の状態を 1 つの終点単語ごとに追って説明する。

(1) 1 巡目処理中の終点単語 a の計算

1 巡目処理中の終点単語 a の計算は、次のようになる。

$$\begin{aligned} a &= 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\ &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\ &= (+1) \end{aligned}$$

入力単語 a は始点単語を持たないので、どのような値の状態をかけても値の状態が確定し、null から (+1) へ変化する。すなわち、これは入力によって値が確定したことを表わしている。その結果、1 巡処理中の単語 a 行完了後の各単語の値の状態は、以下の表 3 のとおりである。

【0 1 4 9】

【表 3】

	a	b	c	d	x_r	e	f	g
値の状態	+1	null	null	null	null	null	null	null

(2) 1 巡目処理中の終点単語 b の計算

1 巡目処理中の終点単語 b の計算は、表 3 の単語の値の状態を用いるので次のようになる。

$$\begin{aligned} b &= 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\ &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \end{aligned}$$

$= (+1)$

入力単語 b も始点単語を持たないので、どのような値の状態をかけても値の状態が確定し、nullから(+1)へ変化する。その結果、1巡処理中の単語 b 行完了後の各単語の値の状態は、以下の表 4 のとおりである。

【 0 1 5 0 】

【表 4】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	null	null	null	null	null	null

例 5 のシステムでは、単語 a と単語 b は画面上の 2 者択一の指示ボタンであるので、どちらか一方が選択されることによって、双方の値の状態が確定することになる。

(3) 1 巡目処理中の終点単語 c の計算

1 巡目処理中の終点単語 c の計算は、表 4 の単語の値の状態を用いるので次のようになる。

$$\begin{aligned}
 & b \\
 & = 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 & = (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 & = (+1)
 \end{aligned}$$

入力単語 c も始点単語を持たないので、どのような値の状態をかけても値の状態が確定し、nullから(+1)へ変化する。その結果、1巡処理中の単語 c 行完了後の各単語の値の状態は、以下の表 5 のとおりである。

【 0 1 5 1 】

【表 5】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	+1	null	null	null	null	null

(4) 1 巡目処理中の終点単語 d の計算

1 巡目処理中の終点単語 d の計算は、表 5 の単語の値の状態を用いるので次のようになる。

$$\begin{aligned}
 & b \\
 & = 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 & = (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 & = (+1)
 \end{aligned}$$

入力単語 d も始点単語を持たないので、どのような値の状態をかけても値の状態が確定し、nullから(+1)へ変化する。その結果、1巡処理中の単語 d 行完了後の各単語の値の状態は、以下の表 6 のとおりである。

【 0 1 5 2 】

【表 6】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	+1	+1	null	null	null	null

(5) 1 巡目処理中の終点単語 x_r の計算

1 巡目処理中の終点単語 x_r の計算は、表 6 の単語の値の状態を用いるので次のようになる。

$$\begin{aligned}
 & x_r \\
 & = 1 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 & = (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1)
 \end{aligned}$$

$= (+1)$

単語 x_r の始点単語となる単語 a 、 b 、 c 、 d の値の状態がすでに確定した、すなわち、表 2 に示した経路作用要素である単語 x_r の定義式実行条件 (1) および (2) が満たされたので、単語 x_r の値の状態も $null$ から確定済みの $(+1)$ へ変化する。その結果、1 巡処理中の単語 x_r 行完了後の各単語の値の状態は、以下の表 7 のとおりである。

【0 1 5 3】

【表 7】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	+1	+1	+1	null	null	null

(6) 1 巡目処理中の終点単語 e の計算

1 巡目処理中の終点単語 e の計算は、表 7 の単語の値の状態を用いるので次のようになる。

x_r

$$\begin{aligned}
 &= 0 \cdot (+1) + 0 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot null + 0 \cdot null + 0 \cdot null \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

単語 e の始点単語となる単語 c 、 d および x_r の値の状態がすでに確定済みであるので、すなわち、定義式 $c + d$ の始点単語の値が決定済み、かつ経路作用要素 x_r が次に実行する基本構造を指定したので、単語 e の値の状態も $null$ から確定済み $(+1)$ へ変化する。その結果、1 巡処理中の単語 e 行完了後の各単語の値の状態は、以下の表 8 のとおりである。

【0 1 5 4】

【表 8】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	+1	+1	+1	+1	null	null

(7) 1 巡目処理中の終点単語 f の計算

1 巡目処理中の終点単語 f の計算は、表 8 の単語の値の状態を用いるので次のようになる。

x_r

$$\begin{aligned}
 &= 0 \cdot (+1) + 0 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot (+1) + 0 \cdot null + 0 \cdot null \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

単語 f の始点単語となる単語 c 、 d および x_r の値の状態がすでに確定済みであるので、単語 f の値の状態も $null$ から $(+1)$ へ変化する。その結果、1 巡処理中の単語 f 行完了後の各単語の値の状態は、以下の表 9 のとおりである。

【0 1 5 5】

【表 9】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	+1	+1	+1	+1	+1	null

例 5 のシステムでは、単語 e と単語 f はそれぞれの実行条件が 2 者択一の条件であるので、どちらか一方の条件が真と確定すれば、他方の条件は偽と確定し、双方の値の状態が確定することになる。たとえば、BS 2 が指定されれば、単語 e の定義式実行条件＝真が確定して単語 e に値が与えられることによって値の状態が確定する。一方、単語 f は、定義式実行条件＝偽が確定して単語 f に値が与えられないことによって値の状態が確定する。従って、経路作用要素がどちらか一方の基本構造を指定したとき、単語 e も単語 f も値

の状態が確定済みとなるのである。

(8) 1 巡目処理中の終点単語 g の計算

1 巡目処理中の終点単語 g の計算は、表 9 の単語の値の状態を用いるので次のようになる。

$$\begin{aligned} x_r &= 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot \text{null} \\ &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\ &= (+1) \end{aligned}$$

単語 g の始点単語となる単語 e および f の値の状態がすでに確定済みであるので、単語 g の値の状態も null から $(+1)$ へ変化する。その結果、1 巡処理中の単語 g 行完了後の各単語の値の状態は、以下の表 10 のとおりである。

【0 1 5 6】

【表 10】

	a	b	c	d	x_r	e	f	g
値の状態	+1	+1	+1	+1	+1	+1	+1	+1

以上で隣接行列 F の総ての単語行の計算処理が 1 巡したことになる。従って、表 10 が $F X$ の結果となる。下記の数式 30 の状態ベクトル X_1 が $F X$ の結果である。

【0 1 5 7】

【数 30】

$$X_1 = \begin{bmatrix} a \\ b \\ c \\ d \\ x_r \\ e \\ f \\ g \end{bmatrix} \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

以上のように、隣接行列 F は、トポロジカルソートによって単語単位プログラムが最適順序で実行されるように並んでいるので、1 巡の計算処理の実行で総ての単語の値の状態が確定し、出力単語 g の値を成立することができた。

I I - 2 - (4) 経路作用要素の排除

経路作用要素の実行条件は、経路作用要素によって指定される基本構造に属する全単語の集合 (1 つ以上からなる) の定義式を実行するための条件と等しい。なぜなら、その基本構造の単語の総ては、その経路作用要素の条件が満されたときに実行されるからである。従って、経路作用要素の実行条件は、経路作用要素によって指定される基本構造に属する単語の条件に移すことができる。

【0 1 5 8】

例 5 のシステムの場合で具体的に述べると次のようになる。まず、経路作用要素である単語 x_r と、単語 e および単語 f の定義式と定義式実行条件は下記の表 11 の通りであった。

【0 1 5 9】

【表 1 1】

単語	定義式	定義式実行条件
x_r	BS 2 を指定	(1) $a = \text{真}$ (ボタン a が押され、 a に値がある) かつ、 $c = \text{真}$ および $d = \text{真}$
	BS 3 を指定	(2) $b = \text{真}$ (ボタン b が押され、 b に値がある) かつ、 $c = \text{真}$ および $d = \text{真}$
e	$c + d$	x_r が BS 2 を指定
f	$d \times d$	x_r が BS 3 を指定

従って、 x_r の条件 (1) 「 $a = \text{真}$ かつ、 $c = \text{真}$ および $d = \text{真}$ 」を、この条件が成立するとき指定する基本構造 BS 2 に属する単語 e の実行条件「 x_r が BS 2 を指定」と置き換え、条件 (2) 「 $b = \text{真}$ かつ、 $c = \text{真}$ および $d = \text{真}$ 」をこの条件が成立するとき指定する基本構造 BS 3 に属する単語 f の実行条件「 x_r が BS 3 を指定」と置き換える、ということになる。この結果、単語 e および単語 f の定義式、定義式実行条件は下記の表 1 2 のようになる。

【0 1 6 0】

【表 1 2】

単語	定義式	定義式実行条件
e	$c + d$	$a = \text{真}$ (ボタン a が押され、 a に値がある) かつ、 $c = \text{真}$ および $d = \text{真}$
f	$c \times d$	$b = \text{真}$ (ボタン b が押され、 b に値がある) かつ、 $c = \text{真}$ および $d = \text{真}$

表 1 2 のように、経路作用要素の実行条件を、指定先の基本構造の単語の実行条件に置き換えると、BS 1 と BS 2 および BS 3 のリンクは、BS 2 および BS 3 の単語の実行条件によって成立するので、経路作用要素は単に、次の基本構造の実行に移る条件が整ったこと (入力単語 c および d に値があり、かつボタン a または b が押された) を示すだけの意味をしか持たなくなる。これは、経路作用要素を取り除いてもシステムの意味に変化が生じないことを意味する。従って、経路作用要素の実行条件を、その条件の成立によって指定される基本構造の単語の実行条件に置き換えを行った後、システム全体の隣接行列から、経路作用要素を除くことができる。

【0 1 6 1】

図 1 4 は、経路作用要素を取り除いて、例 5 のシステムの単語の関係を有向グラフで表したものである。図 1 4 では、経路作用要素である単語 x_r がなくなり、単語 e および単語 f と始点単語との関係は、上記の表 1 2 で定義された定義式と定義式実行条件に従って示されている。すなわち、単語 e は、始点単語として単語 a (定義式実行条件「実行ボタン a が押された」)、単語 c および単語 d (定義式「 $c + d$ 」および、定義式実行条件「 $c = \text{真}$ および $d = \text{真}$ 」) を持つ。単語 f は、始点単語として単語 b (定義式実行条件「実行ボタン b が押された」)、単語 c および単語 d (定義式「 $c + d$ 」および、定義式実行条件「 $c = \text{真}$ および $d = \text{真}$ 」) を持つ。単語 e および単語 f から単語 g への関係は図 1 1 と変わらない。

【0 1 6 2】

経路作用要素を取り除いた例 5 のシステムを、図 1 4 の有向グラフに従って隣接行列に表わせば、以下ようになる。数式 3 1 の $F 1'$ は、基本構造 BS 1 (入力) の隣接行列である。経路作用要素である単語 x_r が要素から削除されている。

【0 1 6 3】

【数 3 1】

$$F1' = \begin{matrix} & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ \mathbf{a} & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{b} & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{c} & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{d} & \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

数式 3 2 の $F 2'$ は、基本構造 $B S 2$ & $B S 3$ の隣接行列である。

【0 1 6 4】

【数 3 2】

$$F2' = \begin{matrix} & \mathbf{e} & \mathbf{f} \\ \mathbf{e} & \begin{bmatrix} 0 & 0 \end{bmatrix} \\ \mathbf{f} & \begin{bmatrix} 0 & 0 \end{bmatrix} \end{matrix}$$

数式 3 3 の $F 3'$ は、基本構造 $B S 1$ (出力) の隣接行列である。

【0 1 6 5】

【数 3 3】

$$F3' = \begin{matrix} & \mathbf{g} \\ \mathbf{g} & [0] \end{matrix}$$

数式 3 4 の $F C 1'$ は、基本構造 $B S 1$ (入力) から基本構造 $B S 2$ & $B S 3$ へ の関係を示す結合隣接行列である。経路作用要素が削除されたので、始点単語の要素として単語 X_r が削除されている。単語 e および単語 f は、表 1 2 に示したように経路作用要素の実行条件を移したことで、それぞれ新たに単語 a 、単語 b を始点単語として持つことになった。従って、終点単語 e は、始点単語 c および d の他に単語 a との交点が 1 となっている。終点単語 f は、始点単語 c および d の他に単語 b との交点が 1 となっている。

【0 1 6 6】

【数 3 4】

$$FC1' = \begin{matrix} & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ \mathbf{e} & \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix} \\ \mathbf{f} & \begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix} \end{matrix}$$

数式 3 5 の $F C 2'$ は、基本構造 $B S 2$ & $B S 3$ から基本構造 $B S 1$ (出力) へ の関係を示す結合隣接行列である。経路作用要素が削除されたので、始点単語の要素として単語 X_r が削除されている。

【0 1 6 7】

【数 3 5】

$$FC2' = g \begin{bmatrix} a & b & c & d & e & f \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

数式 3 6 の $FC3'$ は、基本構造 $BS2$ & $BS3$ から基本構造 $BS1$ (入力) への関係がないことを示す結合隣接行列である。経路作用要素が削除されたので、始点単語の要素として単語 X_r が削除されている。

【0 1 6 8】

【数 3 6】

$$FC3' = \begin{bmatrix} e & f \\ a & 0 & 0 \\ b & 0 & 0 \\ c & 0 & 0 \\ d & 0 & 0 \end{bmatrix}$$

数式 3 7 の $FC4'$ は、基本構造 $BS1$ (出力) から基本構造 $BS1$ (入力) への関係がないことを示す結合隣接行列である。経路作用要素が削除されたので、始点単語の要素として単語 X_r が削除されている。

【0 1 6 9】

【数 3 7】

$$FC4' = \begin{bmatrix} g \\ a & 0 \\ b & 0 \\ c & 0 \\ d & 0 \end{bmatrix}$$

数式 3 8 の $FC5'$ は、基本構造 $BS1$ (出力) から基本構造 $BS2$ & $BS3$ への関係がないことを示す結合隣接行列である。

【0 1 7 0】

【数 3 8】

$$FC5' = \begin{bmatrix} g \\ e & 0 \\ f & 0 \end{bmatrix}$$

システムの隣接行列 F は、経路作用要素の条件の置き換えと経路作用要素の削除を行うと、下記の数式 3 9 に示す隣接行列 F' で表わすことができる。

【0 1 7 1】

【数 39】

$$F' = \begin{bmatrix} F1' & 0 & 0 \\ FC1' & F2' & 0 \\ FC2' & & F3' \end{bmatrix} = \begin{matrix} & \begin{matrix} a & b & c & d & e & f & g \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

上記数式で、「 $F' =$ 」の右辺に示された、 $F1'$ 、 $F2'$ 、 $F3'$ 、 $FC1'$ 、 $FC2'$ と0（行列の交点が総て0である隣接行列で、 $FC3'$ 、 $FC4'$ 、 $FC5'$ に相当する）で構成された行列は、基本構造単位の隣接行列と結合隣接行列がどのように統合されてシステム全体を表わすかを示している。次に、それらの要素を単語に変えたものが、その右辺にある a 、 b 、 c 、 d 、 e 、 f 、 g を行列の要素とする隣接行列である。 $F1'$ 、 $F2'$ 、 $F3'$ 、 $FC1'$ 、 $FC2'$ 、 $FC3'$ 、 $FC4'$ 、 $FC5'$ がどのように統合されて F' を構成するのか、さらにわかりやすく示したのが図15である。図15の1201の部分に $F1'$ 、1202が $F2'$ 、1203が $F3'$ 、1204が $FC1'$ 、1205が $FC2'$ である。行列の交点が総て0である結合隣接行列は、 $FC3'$ が1206、 $FC4'$ が1207、 $FC5'$ が1208を構成する。

【0172】

以上のようにして、2つ以上の基本構造から成る1つのシステムを、経路作用要素を排除して、1つの隣接行列で表わすことができる。

次に、このシステムの隣接行列に対して、トポロジカルソートを行って、総ての単語の値の状態が最少実行回数で確定（すなわち総ての出力単語の生成が完了）するような最適順序に単語単位のプログラムの並び替えを行う。隣接行列 F' をみると、図16に示すように、値が1である始点単語との交点は総て左下三角形（1601）内にある。I章で述べたように、これは単語が最適順序に並んでいることを意味する。従って、数式39の隣接行列 F' は、すでにトポロジカルソート済みであって、初期値の単語の状態ベクトルを与えれば、最少実行回数で出力単語の生成を完了することができる単語の順序になっているといえる。

II-2-(5) 経路作用要素を排除した構造の検証

このセクションでは、前セクションで示した、経路作用要素を排除したシステムの構造を示す隣接行列 F' が、経路作用要素を排除する前の構造である隣接行列 F と同じ関数機能を持つことを検証する。以下に、隣接行列 F' に単語の状態ベクトルを与えて出力単語 g の値を求める過程を示す。m回目の計算処理の結果、単語の状態ベクトル X_m の値が総て確定済みとなれば、 F' も F と同等に機能する関数だと言える。また、トポロジカルソートを施したことによって、最少実行回数で出力単語の生成を完了できることも検証する。例5のシステムである F' も、1巡処理で単語 g の値の状態が確定すれば、最少実行回数で実行できたと言える。

前述したように、単語の状態ベクトルの初期値は総ての要素が「null（未決定）」であるので、 F' に最初に与えるべき単語の状態ベクトル X' は下記の数式40のようになる。

【 0 1 7 3 】

【 数 4 0 】

$$X' = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{bmatrix} = \begin{bmatrix} \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \\ \text{null} \end{bmatrix}$$

数式 3 9 の隣接行列 F' (すなわち例 5 のシステム) に、数式 3 9 の状態ベクトル X' をかける計算処理を施す (すなわち、例 5 のシステムに入力を行う)。 $F' X'$ の計算処理によって変化する単語の値の状態を 1 つの終点単語ごとに追って説明する。

(1) F' 1 巡目処理中の終点単語 a の計算

終点単語 a は、計算処理に X' の単語の値の状態を用いるので、1 巡目処理中の終点単語 a の計算は、次のようになる。

$$\begin{aligned} a &= 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\ &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\ &= (+1) \end{aligned}$$

入力単語 a は始点単語を持たないので、値の状態が確定し、 null から $(+1)$ へ変化する。すなわち、これは入力によって値が確定したことを表わしている。その結果、1 巡処理中の単語 a 行完了後の各単語の値の状態は、以下の表 1 3 のとおりである。

【 0 1 7 4 】

【表 1 3】

	a	b	c	d	e	f	g
値の状態	+1	null	null	null	null	null	null

(2) F' 1 巡目処理中の終点単語 b の計算

終点単語 b は、計算処理に表 1 3 の単語の値の状態を用いるので、1 巡目処理中の終点単語 b の計算は、次のようになる。

$$\begin{aligned} b &= 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\ &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\ &= (+1) \end{aligned}$$

入力単語 b も始点単語を持たないので、値の状態が確定し、 null から $(+1)$ へ変化する。その結果、1 巡処理中の単語 a 行完了後の各単語の値の状態は、以下の表 1 4 のとおりである。

【 0 1 7 5 】

【表 1 4】

	a	b	c	d	e	f	g
値の状態	+1	+1	null	null	null	null	null

(3) F' 1 巡目処理中の終点単語 c の計算

終点単語 c は、計算処理に表 1 4 の単語の値の状態を用いるので、1 巡目処理中の終点単語 c の計算は、次のようになる。

c

$$\begin{aligned}
 &= 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

入力単語 c も始点単語を持たないので、値の状態が確定し、nullから(+1)へ変化する。その結果、1巡処理中の単語 c 行完了後の各単語の値の状態は、以下の表 1 5 のとおりである。

【0 1 7 6】

【表 1 5】

	a	b	c	d	e	f	g
値の状態	+1	+1	+1	null	null	null	null

(4) F' 1巡目処理中の終点単語 d の計算

終点単語 d は、計算処理に表 1 5 の単語の値の状態を用いるので、1巡目処理中の終点単語 d の計算は、次のようになる。

$$\begin{aligned}
 &d \\
 &= 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

入力単語 d も始点単語を持たないので、値の状態が確定し、nullから(+1)へ変化する。その結果、1巡処理中の単語 d 行完了後の各単語の値の状態は、以下の表 1 6 のとおりである。

【0 1 7 7】

【表 1 6】

	a	b	c	d	e	f	g
値の状態	+1	+1	+1	+1	null	null	null

(5) F' 1巡目処理中の終点単語 e の計算

終点単語 e は、計算処理に表 1 6 の単語の値の状態を用いるので、1巡目処理中の終点単語 e の計算は、次のようになる。

$$\begin{aligned}
 &e \\
 &= 1 \cdot (+1) + 0 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

単語 e の始点単語である単語 a、c、d の値の状態が確定しているので、単語 e の値の状態もnullから(+1)へ変化して確定する。その結果、1巡処理中の単語 e 行完了後の各単語の値の状態は、以下の表 1 7 のとおりである。

【0 1 7 8】

【表 1 7】

	a	b	c	d	e	f	g
値の状態	+1	+1	+1	+1	+1	null	null

(6) F' 1巡目処理中の終点単語 f の計算

終点単語 f は、計算処理に表 1 7 の単語の値の状態を用いるので、1巡目処理中の終点単語 f の計算は、次のようになる。

f

$$\begin{aligned}
 &= 0 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot (+1) + 0 \cdot \text{null} + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

単語 f の始点単語である単語 b、c、d の値の状態が確定しているので、単語 f の値の状態も null から (+1) へ変化して確定する。その結果、1 巡処理中の単語 f 行完了後の各単語の値の状態は、以下の表 1 8 のとおりである。

【0 1 7 9】

【表 1 8】

	a	b	c	d	e	f	g
値の状態	+1	+1	+1	+1	+1	+1	null

(7) F' 1 巡目処理中の終点単語 g の計算

終点単語 g は、計算処理に表 1 8 の単語の値の状態を用いるので、1 巡目処理中の終点単語 g の計算は、次のようになる。

$$\begin{aligned}
 &g \\
 &= 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 0 \cdot (+1) + 1 \cdot (+1) + 1 \cdot (+1) + 0 \cdot \text{null} \\
 &= (+1) + (+1) + (+1) + (+1) + (+1) + (+1) + (+1) \\
 &= (+1)
 \end{aligned}$$

単語 g の始点単語である単語 e、f の値の状態が確定しているので、単語 g の値の状態も null から (+1) へ変化して確定する。その結果、1 巡処理中の単語 g 行完了後の各単語の値の状態は、以下の表 1 9 のとおりである。

【0 1 8 0】

【表 1 9】

	a	b	c	d	e	f	g
値の状態	+1	+1	+1	+1	+1	+1	+1

以上で隣接行列 F' の総ての単語行の計算処理が 1 巡したことになる。従って、表 1 9 の単語の値の状態が F' X' の結果となる。下記の数式 4 1 の状態ベクトル X' ₁ が F' X' の結果である。

【0 1 8 1】

【数 4 1】

$$X'_1 = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{bmatrix} = \begin{bmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

以上のように、隣接行列 F' は、トポロジカルソートによって単語単位プログラムが最適順序で実行されるように並んでいるので、1 巡の計算処理の実行で総ての単語値が確定し、出力単語 g の値を成立することができた。

また、経路作用要素の単語を除いた隣接行列 F' も、状態ベクトルの初期値を与える（入力を行う）ことで、隣接行列 F と同様に総ての単語の値の状態が確定し、出力単語 g の値が生成できたことがわかる。従って、経路作用要素のあるシステムである隣接行列 F と、経路作用要素のないシステムである隣接行列 F' は、システムとして同等の処理機能を

持つことが検証できた。

上記で述べたように、全体システムが複数の基本構造を有する場合に、これを統合し、さらに当該統合されたものに対して（例えばトポロジカルソートのような）探索を施すことで、L y e e（登録商標）等のソフトウェアの自律的開発技術の適用に際して各段のパフォーマンスを提供する。これが例えば、統合されずに複数の基本構造をそのまま有する場合には、本願に係る発明によるほどのパフォーマンスは得られず、また、トポロジカルソート等の探索技法によって整序しなかった場合には本願に係る発明によるほどのパフォーマンスは得られない。

I I I 章 むすび

処理経路図を作成すること、すなわち分割してとらえることは、システムの要件を正確にとらえるために非常に有用である。処理経路図を作成することは、出力条件に共通要素を持つ単語を論理レコードの単位でとらえることである。条件の共通要素を経路作用要素に割り当てることは、効率的にユーザ要件を定義するためには大きなメリットがある。

一方、プログラムコードを生成し、実行する時点では、基本構造に分割したシステムを統合することが有用である。なぜなら、システム全体の単語単位のプログラムの実行順序を、最少実行回数で出力単語の生成を完了できる順序に並べることができるからである。

本明細書では、単語単位のプログラムからなるシステムにおける、出力データを生成するための単語単位プログラムの処理を、無駄な繰り返しを排除して最少実行回数で完了することを課題として、以下の手段で解決することについて述べた。

1) 同じ出力条件を持つ単語の集合（論理レコード）を、1つの基本構造として、システム全体を処理経路図でとらえる。これによって、同じ基本構造に属する単語単位のプログラムに共通する実行条件（経路作用要素がその基本構造を指定する条件）を的確にとらえることができる。

2) 経路作用要素も単語の1つみなし、単語の関係を定義している単語単位のプログラムとして、要件を定義する。すなわち、どのような条件が成立したとき（定義式実行条件）、どの基本構造の実行を指定するか（定義式）、を定義する。

3) この経路作用要素の定義式実行条件を、その条件が成立するときに経路作用要素が指定する基本構造に属する単語の定義式実行条件に置くことによって、経路作用要素を排除する。経路作用要素の実行条件を、指定先の基本構造の単語の実行条件に置くことによって、経路作用要素を削除しても、基本構造間のリンク（すなわち単語間のリンク）が成立するので、システム全体を1つにまとめることが可能になる。

4) こうして得られた単一化されたシステム全体の単語単位のプログラム群（経路作用要素を含めない）に対して、トポロジカル・ソートを行って単語単位のプログラム群を最適順序に並び替える。これにより、無駄な繰り返しを回避し、最少実行回数でプログラムを実行することが可能となる。

本発明は当該技術分野における通常の知識を有する者にとって修正および改変が容易に数多くなし得るので、図示および記述されたものと寸分違わぬ構成および動作に本発明を限定することは望ましくないことであり、従って、あらゆる適切な改変体および等価体は本発明の範囲に含まれるものと見なされうる。前述の本発明に係る実施の具体的形態の説明および例示によって詳細に記述されたが、本願の特許請求の範囲のみならず本発明に係る開示事項全体に定義された本発明の範囲から逸脱することなしに、修正、置換、および

、変更が数多く可能である。

【0182】

また、本願に係る発明は、その適用において、上記の記述において説明されるか、或いは、図面に示された要素の詳細な解釈及び組み合わせに限定されるものではない。本発明は、他の実施形態が可能であり、種々の方法で実用および実施可能である。また、ここで用いられた語法および用語は記述を目的とするものであり、限定的に働くものとみなされてはならない。

【0183】

従って、当該技術分野における通常の知識を有する者は、本開示の基調となる概念は、本発明の幾つかの目的を実施するための他の構造、方法、及び、システムを設計するための基礎として容易に利用され得ることを理解するはずである。従って、本発明の趣旨および範囲から逸脱しない限り、本願の特許請求の範囲にはそのような等価な解釈が含まれるものと見なされるものである。

【0184】

上記の詳細な説明では、有向グラフ表記、隣接行列算出、トポロジカル・ソート、並び替えという順序で本発明の思想を実現する方法について述べた。しかしこれらはどれも絶対不可欠な要素というのではなく、例えば、有向グラフ表記の表記を経ないで直接隣接行列算出→トポロジカル・ソート→並び替えという順序で本発明の思想を実現してもよい。

【0185】

さらに、有向グラフに対して、例えば幅優先探索 (Breadth-First Search)、反復深化法 (Iterative Deeping)、発見的探索 (Heuristic Search)、ヒル・クライミング (Hill Climbing)、最適優先探索 (Best-First Search)、オイラー (Euler) の一筆書き、ダイクストラ (Dijkstra) 法等の各種探索技法を組み合わせることで並び替えてもよい。

【0186】

或いは、隣接行列算出のあとを、これらの各種探索技法を組み合わせることで本発明の根本思想を実現してもよい。

【0187】

また、本発明に係る技術思想は、例えばコンピュータソフトウェアの自動開発装置、自動開発プログラム、自動開発プログラムを記録した記録媒体、伝送媒体、紙媒体としても、また、自動開発プログラムを登載したコンピュータ・装置、自動開発プログラムを実行するクライアント・サーバ形式等といったカテゴリーにおいても実現、利用可能であることはいうまでもない。

【0188】

さらに本発明は、単一プロセッサ、単一ハードディスクドライブ、及び、単一ローカルメモリを備えたコンピュータシステムに限らず、当該システムのオプションとして、任意の複数または組み合わせプロセッサ又は記憶デバイスを装備するにも適している。コンピュータシステムは、精巧な計算器、掌タイプコンピュータ、ラップトップ/ノートブックコンピュータ、ミニコンピュータ、メインフレームコンピュータ、及び、スーパーコンピュータ、ならびに、これらの処理システムネットワーク組合わせを含む。本発明の原理に従って作動する任意の適切な処理システムによって代替されうるし、また、これらと組合せて用いることも可能である。

【0189】

また、本発明に係る技術思想は、もとよりあらゆる種類のプログラミング言語に対応可能である。さらに本発明に係る技術思想は、あらゆる種類・機能のアプリケーションソフトウェアに対しても適用可能である。

【0190】

またさらに本願発明は、その技術思想の同一及び等価に及ぶ範囲において様々な変形、

追加、置換、拡大、縮小等を許容するものである。また、本願発明を用いて生産されるソフトウェアが、その2次的生産品に登載されて商品化された場合であっても、本願発明の価値は何ら減ずるものではない。

【産業上の利用可能性】

【0191】

上記で規定される本発明では、繰返しをなくす前処理を自動で行い、こうして得られた前処理済の単語単位のプログラムをもとにL y e e（登録商標）方法論により自動的に目的プログラム生成を行う。つまり、ユーザ要件の洗練化（整列）から目的プログラムの生成に至るまで自動化することが可能となる。これにより、ソフトウェア生産の大幅な効率向上、生産性向上、品質向上等、ソフトウェア産業上に大きな効果をもたらす。

【図面の簡単な説明】

【0192】

【図1】状態単語 α についての数式 3 を計算するプログラムの構造を示したフローチャートである。

【図2】 $a \leq b$ (aからbが作られる)という関係をもつ順序を説明するための有向グラフの例である。

【図3】有向グラフの例である。

【図4】あるノード1乃至8が形成する特定の半順序関係を表した有向グラフである。

【図5】トポロジカル・ソートの動作を説明するためのフローチャートである。

【図6】L y e e（登録商標）方法論における単語単位の論理要素をノードとした場合にトポロジカル・ソートを用いて繰返しをなくすための自動制御の動作を説明するためのフローチャートである。

【図7】本発明をソフトウェア開発装置として実現する場合の機能構成を示したブロック図である。

【図8】処理経路図の例である。

【図9】例5のシステムの画面の説明図である。

【図10】例5のシステムの単語を基本構図単位に示した説明図である。

【図11】例5のシステムの単語の関係を表わした有向グラフである。

【図12】例5のシステム全体を1つの隣接行列Fで表わす過程を説明するための図である。

【図13】隣接行列Fがトポロジカルソート済みであることを説明するための図である。

【図14】例5のシステムの構造から経路作用要素を削除したときの、単語の関係を表わした有向グラフである。

【図15】例5のシステムの構造から経路作用要素を削除したとき、システム全体を1つの隣接行列F'で表わす過程を説明するための図である。

【図16】隣接行列F'がトポロジカルソート済みであることを説明するための図である。

【符号の説明】

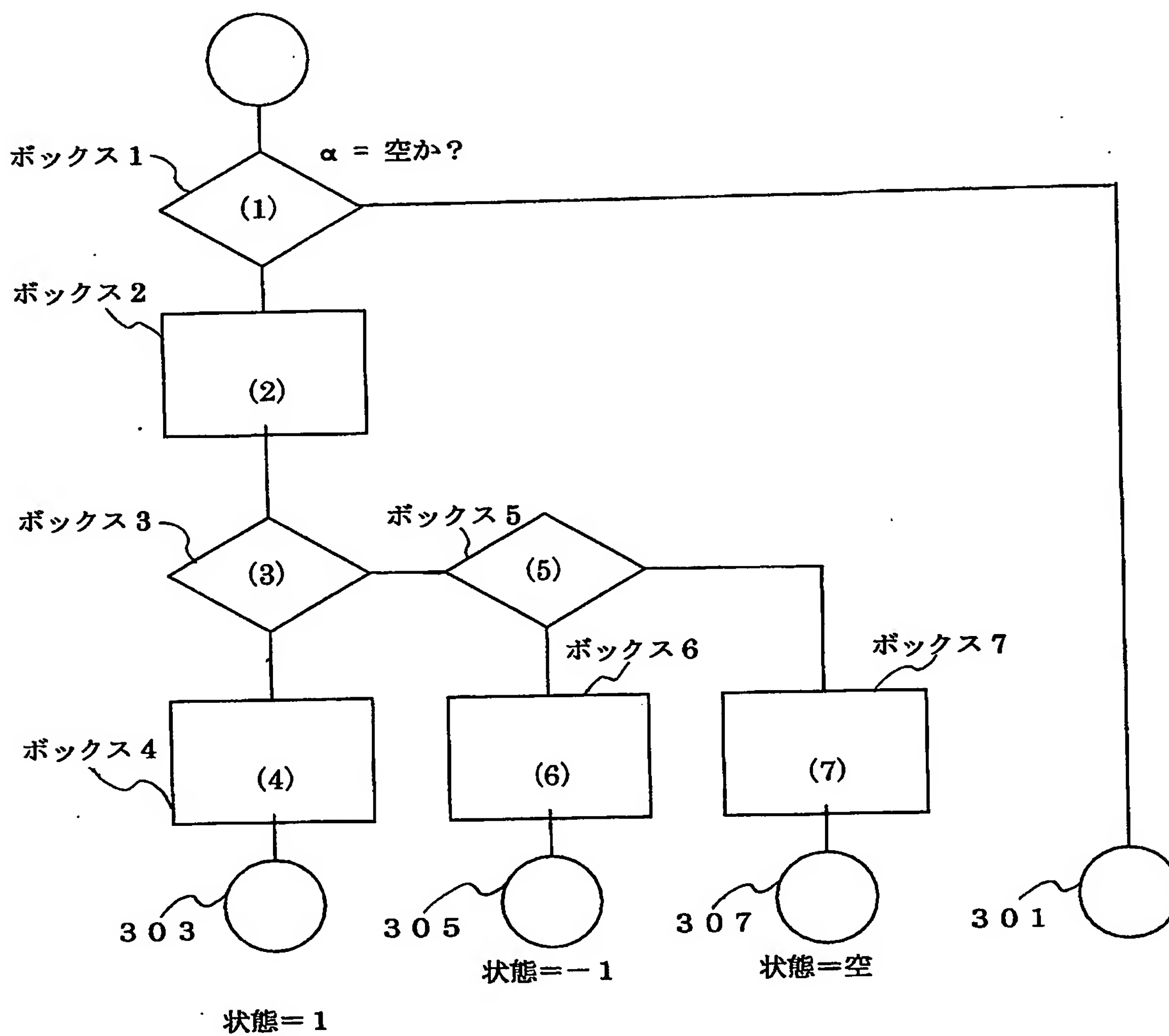
【0193】

- 1101 基本構造BS1（入力）
- 1102 基本構造BS2
- 1103 基本構造BS3
- 1104 基本構造BS1（出力）
- 1201 隣接行列F1'
- 1202 隣接行列F2'
- 1203 隣接行列F3'
- 1204 結合隣接行列FC1'

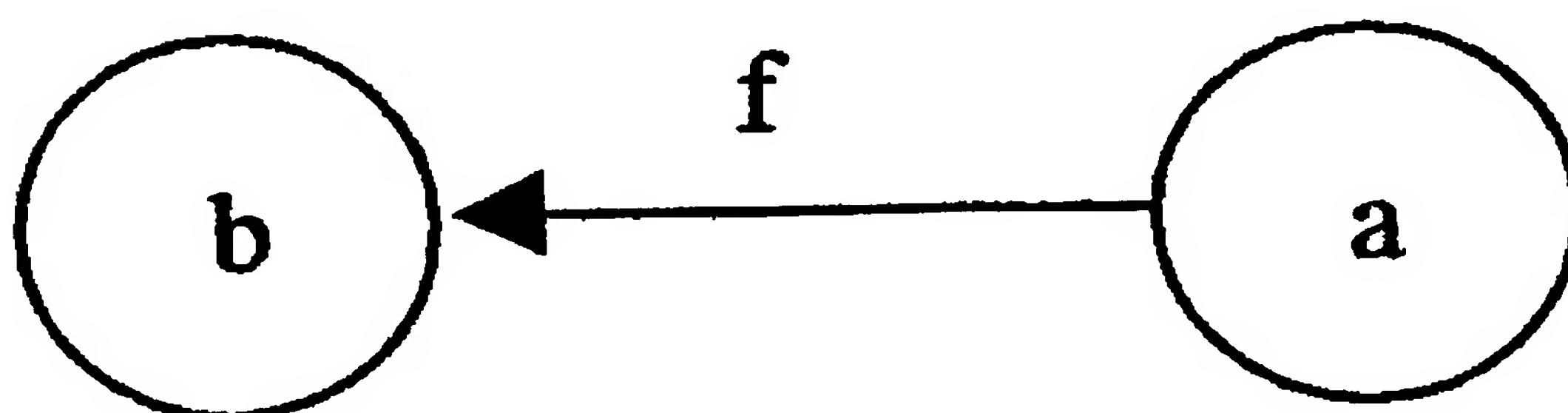
1 2 0 5 結合隣接行列 F C 2'

【書類名】 図面

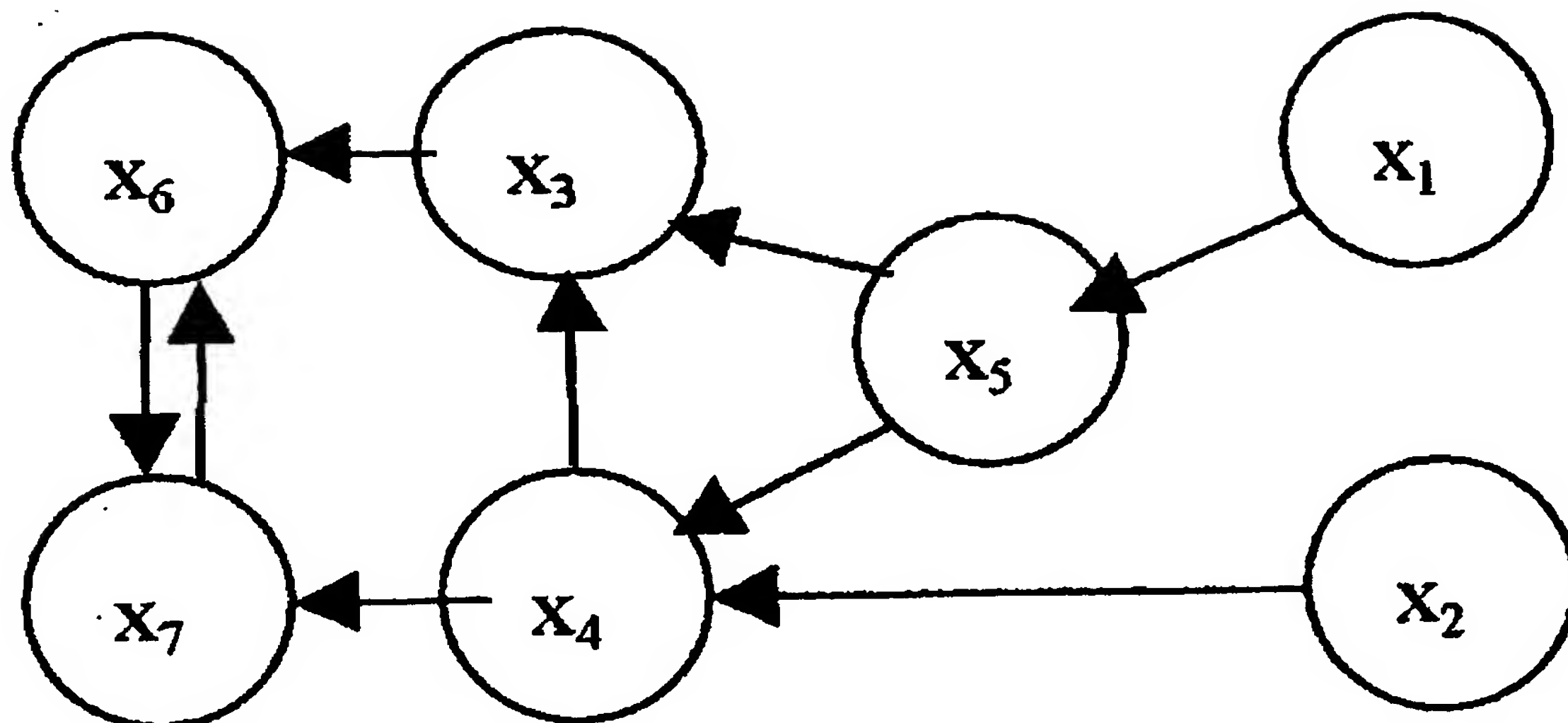
【図 1】



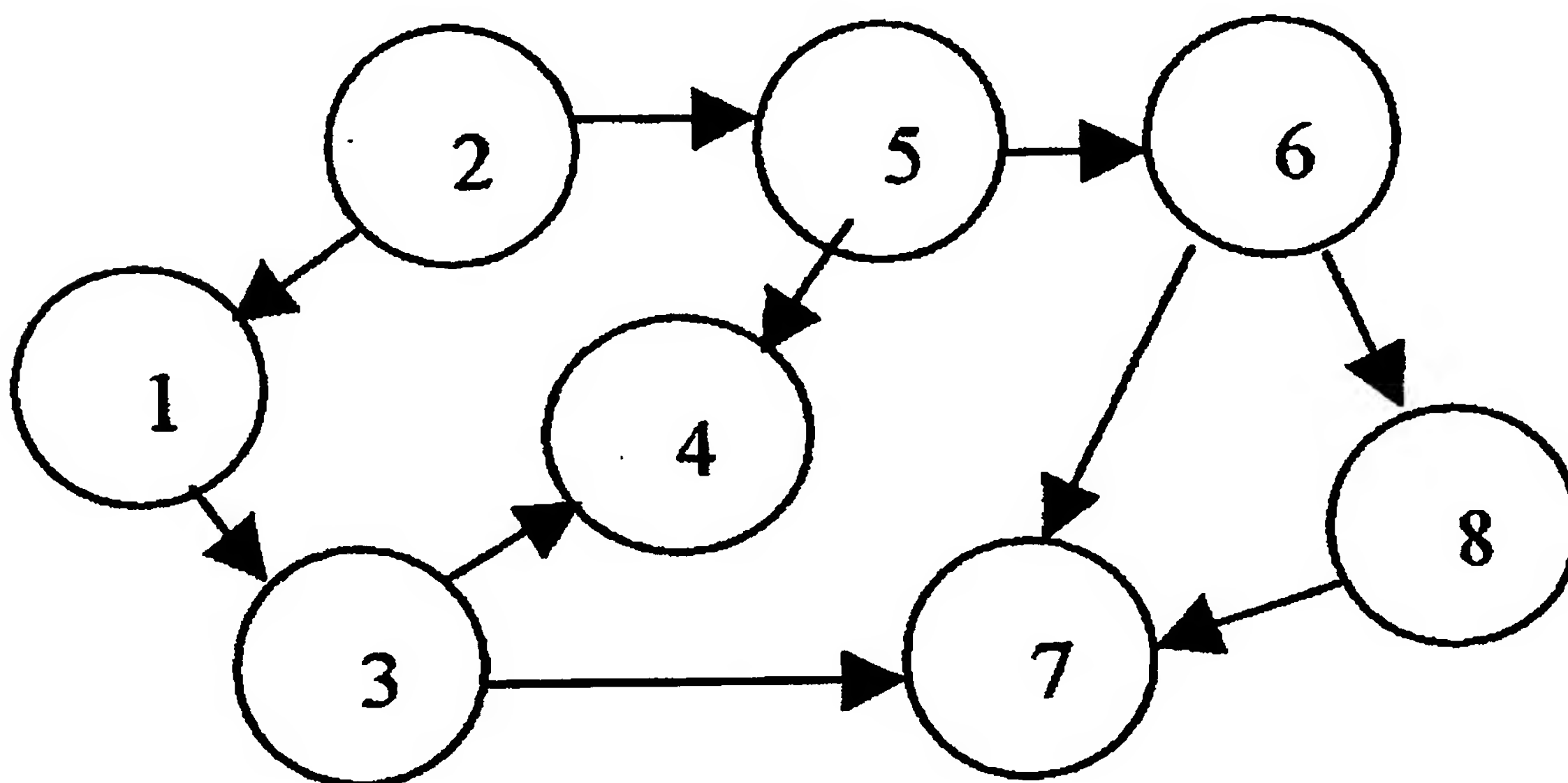
【図 2】



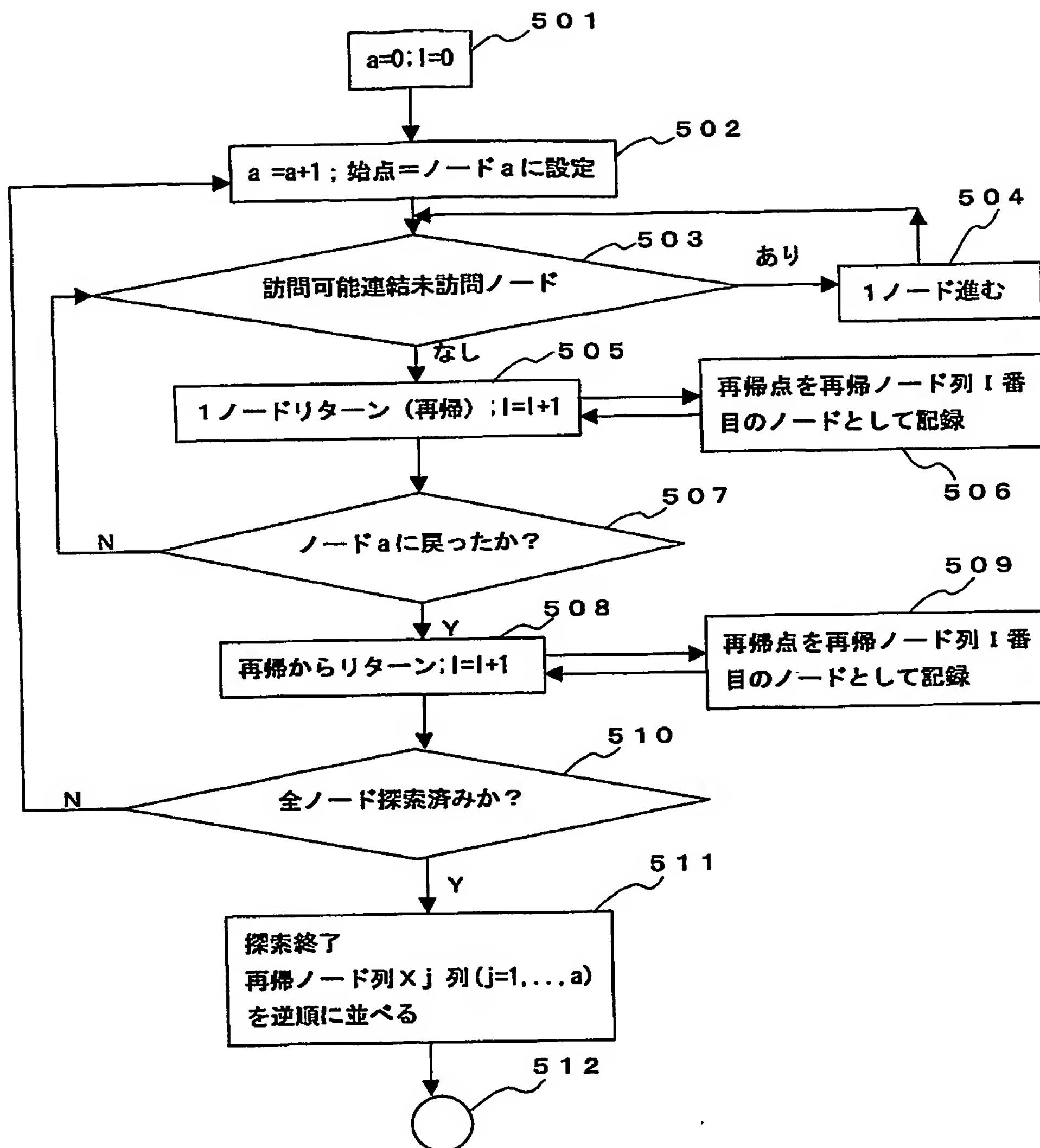
【図 3】



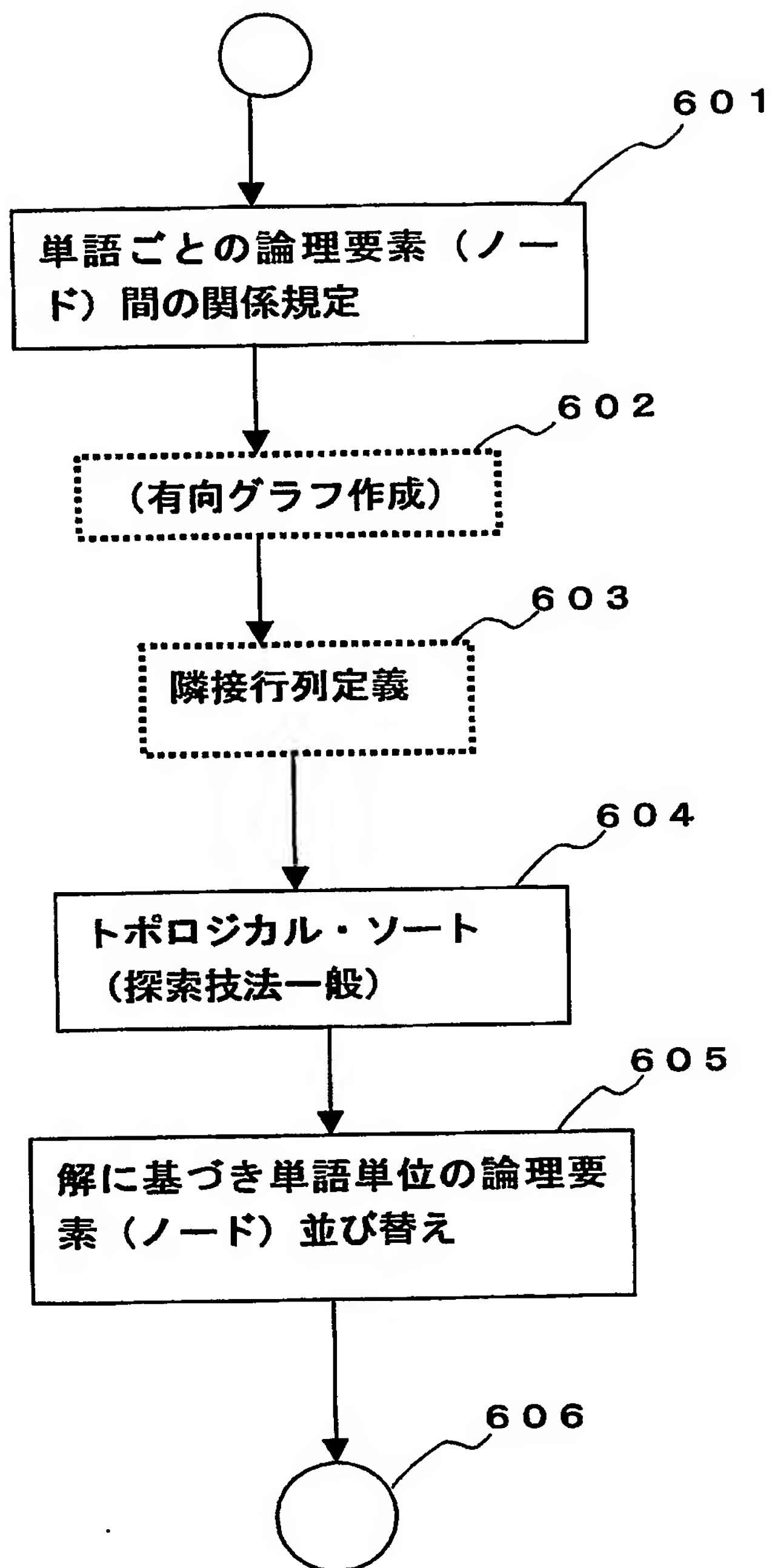
【図 4】



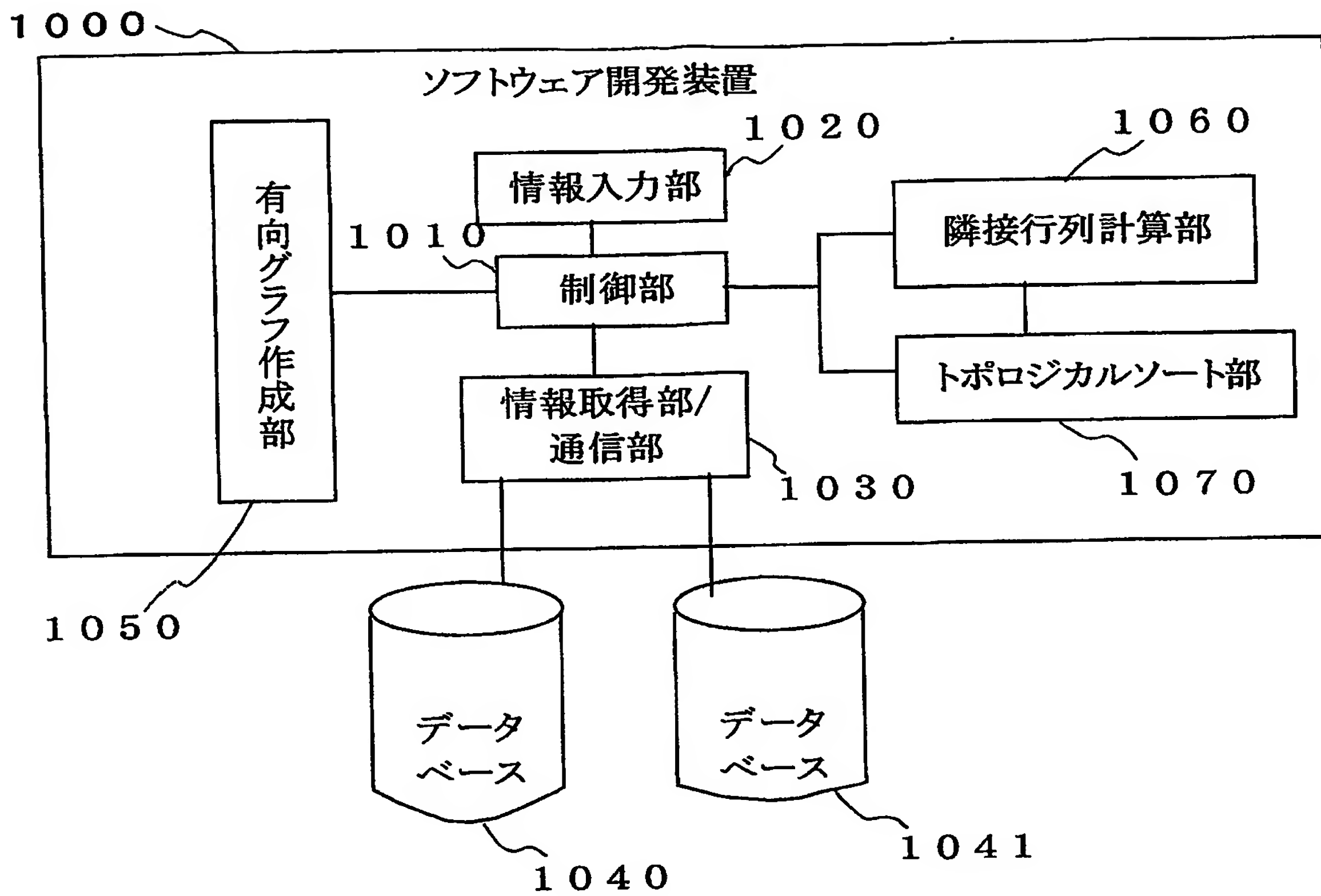
【図 5】



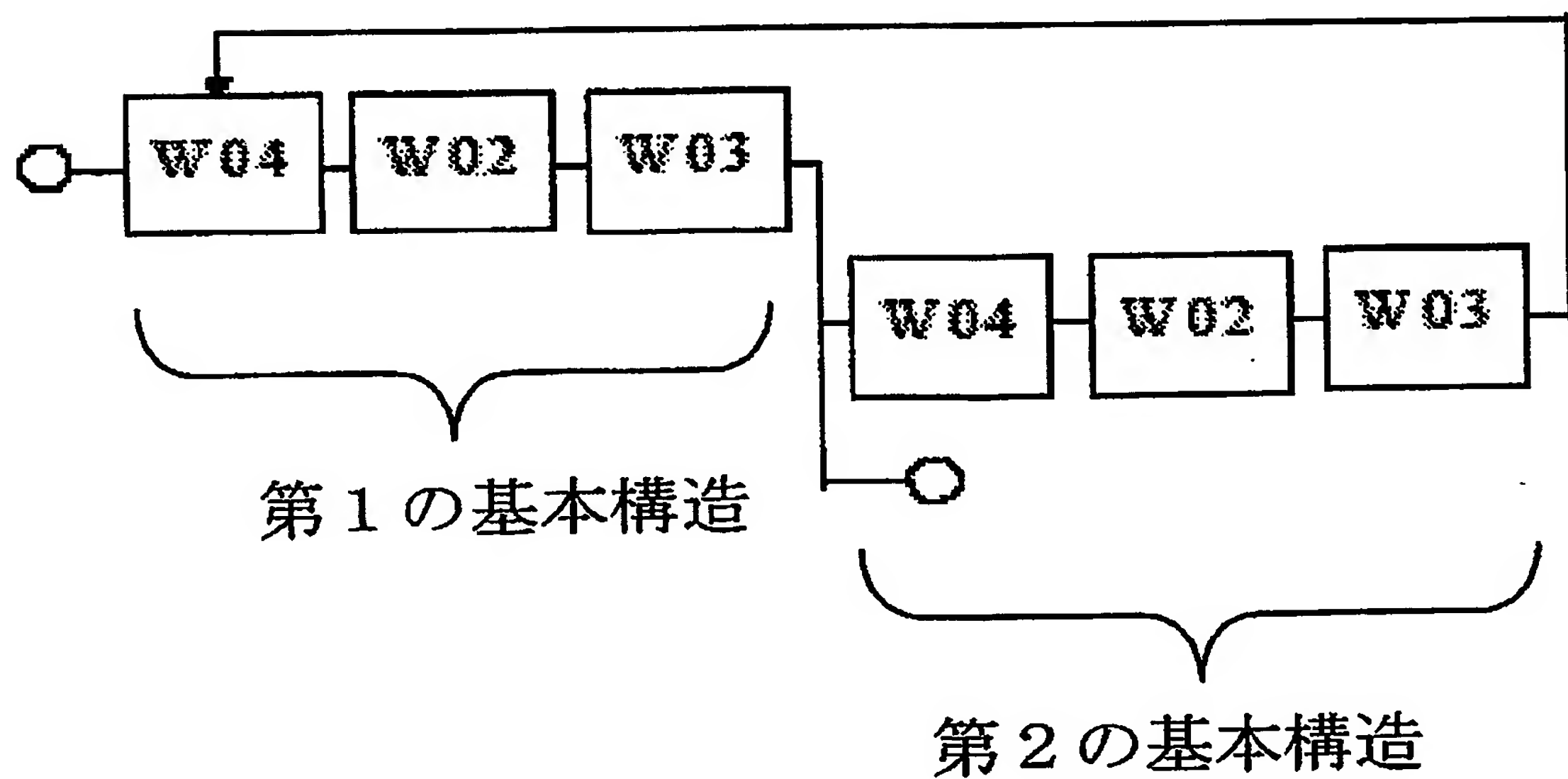
【図 6】



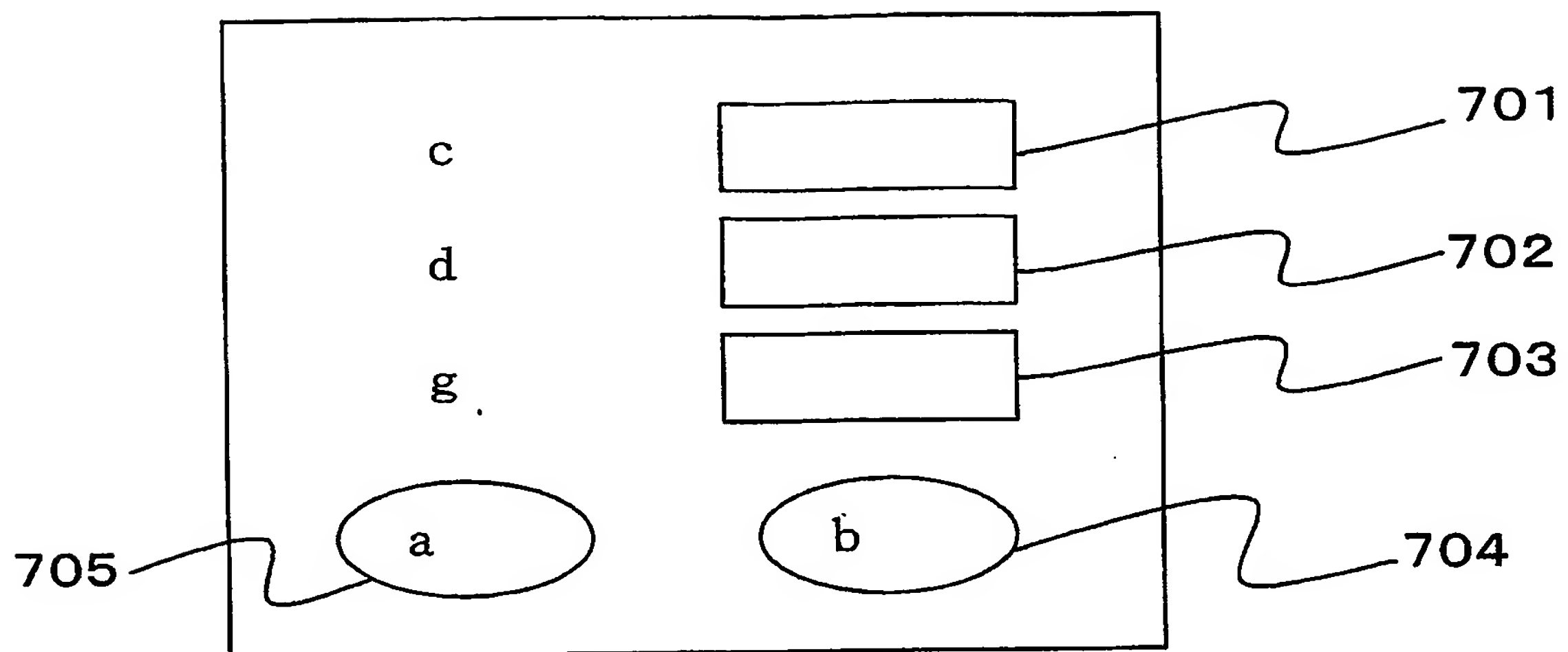
【図 7】



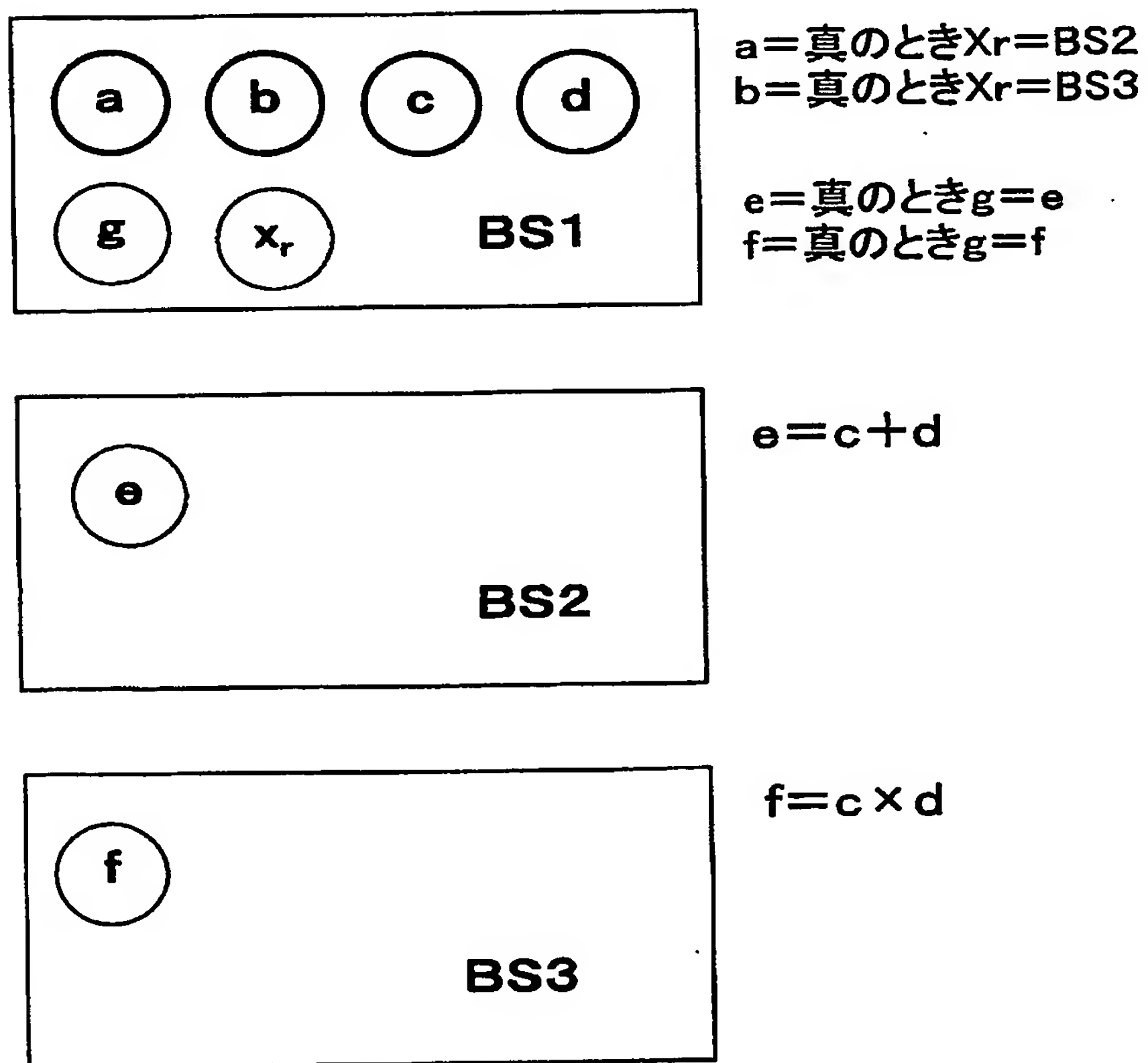
【図 8】



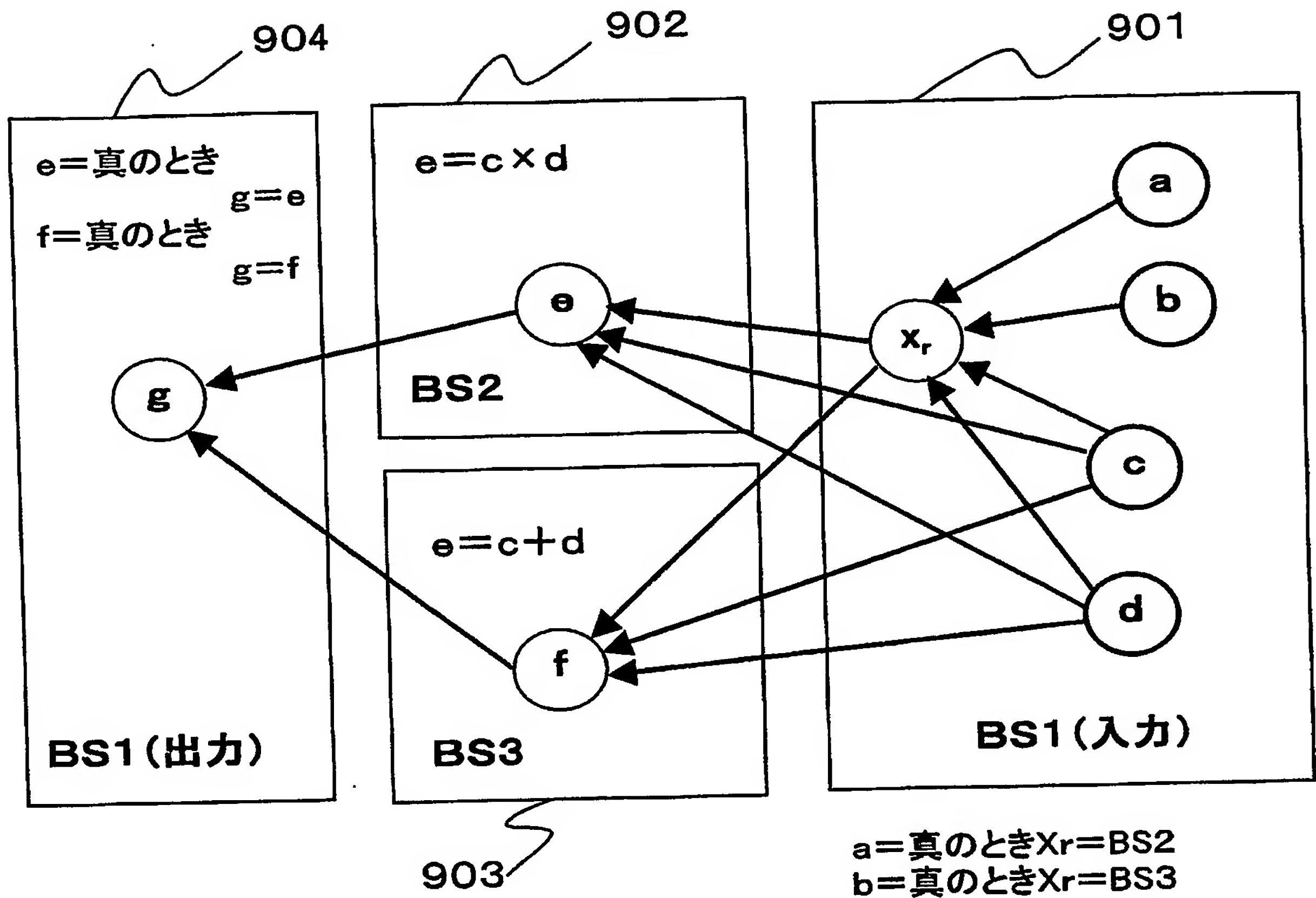
【図 9】



【図 10】



【図 1 1】



【図 1 2】

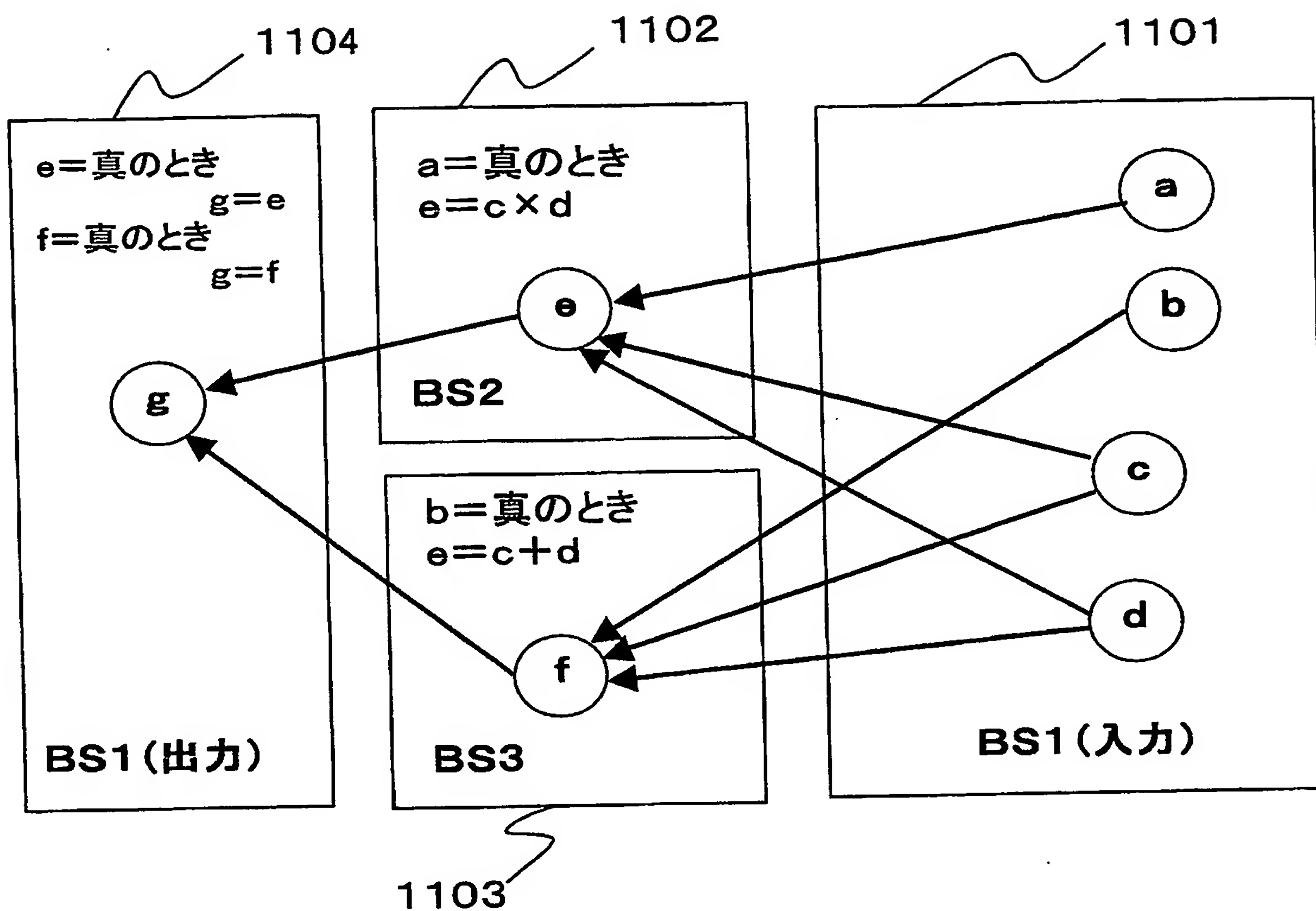
	a	b	c	d	x_r	e	f	g
a	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0
c	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0
x_r	1	1	1	1	0	0	0	0
e	0	0	1	1	1	0	0	0
f	0	0	1	1	1	0	0	0
g	0	0	0	0	0	1	1	0

【図 13】

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>x_r</i>	<i>e</i>	<i>f</i>	<i>g</i>
<i>a</i>	0	0	0	0	0	0	0	0
<i>b</i>	0	0	0	0	0	0	0	0
<i>c</i>	0	0	0	0	0	0	0	0
<i>d</i>	0	0	0	0	0	0	0	0
<i>x_r</i>	1	1	1	1	0	0	0	0
<i>e</i>	0	0	1	1	1	0	0	0
<i>f</i>	0	0	1	1	1	0	0	0
<i>g</i>	0	0	0	0	0	1	1	0

1 3 0 1

【図 14】



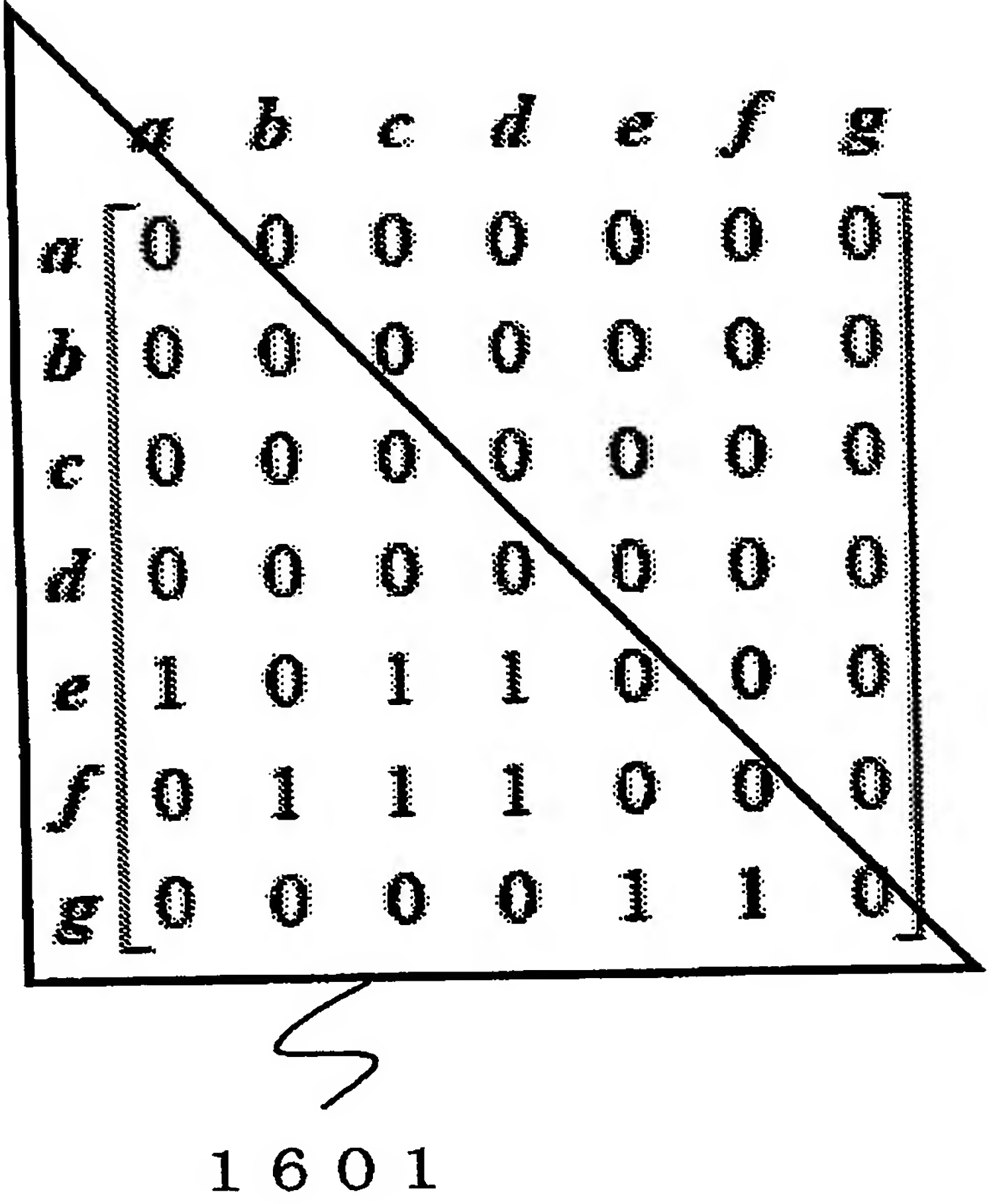
【図 15】

Figure 15 shows a truth table with 7 rows and 7 columns. The columns are labeled a, b, c, d, e, f, g . The rows are labeled a, b, c, d, e, f, g . The table is divided into several regions by brackets:

- Region 1201: The top 4 rows (a, b, c, d) and the first 4 columns (a, b, c, d).
- Region 1202: The top 4 rows (a, b, c, d) and the last 3 columns (e, f, g).
- Region 1203: The bottom 3 rows (e, f, g) and the last 3 columns (e, f, g).
- Region 1204: The bottom 3 rows (e, f, g) and the first 4 columns (a, b, c, d).
- Region 1205: The bottom 3 rows (e, f, g) and the first 4 columns (a, b, c, d).
- Region 1206: The top 4 rows (a, b, c, d) and the last 3 columns (e, f, g).
- Region 1207: The top 4 rows (a, b, c, d) and the last 3 columns (e, f, g).
- Region 1208: The bottom 3 rows (e, f, g) and the last 3 columns (e, f, g).

	a	b	c	d	e	f	g
a	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0
c	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0
e	1	0	1	1	0	0	0
f	0	1	1	1	0	0	0
g	0	0	0	0	1	1	0

【図 16】



【書類名】 要約書**【要約】****【課題】**

単語単位のプログラムからなるシステムにおける、出力データを生成するための単語単位プログラムの処理を、無駄な繰り返しを排除して最少実行回数で完了すること。

【解決手段】

経路作用要素も単語の 1 つみなし、単語の関係を定義している単語単位のプログラムとして、要件を定義する。この経路作用要素の定義式実行条件を、その条件が成立するときに経路作用要素が指定する基本構造に属する単語の定義式実行条件に置くことによって経路作用要素を排除し、システム全体の構造を 1 つにまとめる。こうして得られた単一化されたシステム全体の単語単位のプログラム群（経路作用要素を含めない）に対して、トポロジカルソートを行って単語単位のプログラム群を最適順序に並び替える。これにより、例えば無駄な繰り返しを回避することができる。

【選択図】 図 1 4

特願 2 0 0 3 - 3 4 6 4 4 2

出 願 人 履 歴 情 報

識別番号

[3 9 6 0 2 3 3 6 2]

1. 変成年月日

1 9 9 6 年 1 0 月 1 6 日

[変更理由]

新規登録

住 所

東京都江東区潮見二丁目 1 0 番 2 4 号

氏 名

カテナ株式会社